HELSINKI UNIVERSITY OF TECHNOLOGY

Department of Electrical and Communications Engineering

Janne Kallio

VERIFICATION OF COMMUNICATIONS PLUG-IN UNIT HARDWARE USING A LOGIC SIMULATOR

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering, Espoo May 28th 2002.

Supervisor

Professor Raimo Sepponen

Instructor

M.Sc. Ari Hurtta

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Janne Kallio

Työn nimi: Tietoliikennepistoyksikön laitteiston verifiointi logiikkasimulaattorilla

Päivämäärä: 28.5.2002 Sivumäärä: 65

Osasto: Sähkö- ja Tietoliikennetekniikka

Professuuri: S-66 Sovellettu elektroniikka

Työn valvoja: Professori Raimo Sepponen

Työn ohjaaja: Dipl.ins. Ari Hurtta

Tietoliikennepistoyksiköiden kompleksisuus kasvaa jatkuvasti. Samoin kasvaa myös niiden verifiointiin käytettävä aika. Simuloinnilla pystytään verifioimaan suunniteltu pistoyksikkö suunnittelun aikana ennen kuin prototyypit valmistuvat. Tällä tavoin pystytään lyhentämään verifiointiin käytettävää aikaa ja virheet löydetään ajoissa ennen kuin ne aiheuttavat suuria kustannuksia. Diplomityön tavoitteena on tutkia miten simuloinnit pitäisi toteuttaa pistoyksikkösuunnittelussa. Tutkittavat simuloinnit sisältävät digitaalisen laitteiston toiminnallisen simuloinnin sekä laitteiston ja ohjelmiston yhteissimuloinnin.

Tässä diplomityössä on ensin tutustuttu elektroniikan ja ohjelmiston suunnitteluprosesseihin sekä suunniteltaviin pistoyksiköihin ja verifiointiteoriaan. Seuraavaksi on käyty läpi käytettävät työkalut ja simulointimallit. Simulointien vuokaavio on toteutettu työssä esiteltyjen simulointien pohjalta. Työn aikana tehdyt simuloinnit tarjosivat myös kuvan siitä mitä kaikkea pystytään simuloimaan.

Suunnitteluryhmässä tarvitaan yksi henkilö, joka valmistelee simulointiympäristön. Yhteistyö suunnittelijoiden kanssa simulointien kaikissa vaiheissa on välttämätöntä. Myös simulointityökalujen ja mallien tuki on tärkeää simulointien onnistumiselle. Jotta simulointeja voidaan tehdä suunniteltaville yksiköille, pitää komponenttien käyttöönoton yhteydessä hankkia myös simulointimallit.

Laitteistosimulointien laajuuden rajoitteena on simulointiin käytettävät resurssit ja aika sekä simulointimallit. Näiden perusteella pitää tapauskohtaisesti päättää mitä simuloidaan. Laitteiston ja ohjelmiston yhteissimulointi soveltuu ainakin prosessorilohkoille ja sillä voidaan verifioida käynnistysohjelmiston toiminta.

Avainsanat: Funktionaalinen simulointi, laitteiston verifiointi, HW/SW yhteissimulointi, digitaalielektroniikka, logiikkasimulaattori.

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author: Janne Kallio

Name of the Thesis: Verification of Communications Plug-in Unit Hardware Using a Logic Simulator

Date: 28.5.2002 Number of Pages: 65

Department of Electrical and Communications Engineering

Professorship: S-66 Applied Electronics

Supervisor: Professor Raimo Sepponen

Instructor: M.Sc. Ari Hurtta

The complexity of communications plug-in units is growing. At the same time verification of the plug-in units is taking more and more time. The design can be verified before the prototypes are ready in the design phase using simulation. This reduces the time consumed in verification and errors can be found before the cost of correction increases significantly. The purpose of this Master's Thesis was to study how simulations should be carried out in plug-in unit design process. Simulations include digital HW simulations and HW/SW co-simulations.

The first part of this Master's Thesis includes the hardware and software design processes and also plug-in units and verification theory is described. Next the tools and simulation models will be introduced. Simulation flow is based on the case studies done during the writing of this Master's Thesis. These case studies also showed which parts are feasible for simulation.

Simulations require one person in the design team, who will set-up the simulation environment. Co-operation with hardware designer is necessary in every phase of the simulation flow. Good support for the models and tools is important for the simulation. Models for the simulation should be acquired when components are taken into use to be able to simulate all plug-in units that are designed.

The limitations in HW simulations are simulation models, resources and time. These will determine the scale of the simulations. HW/SW co-simulations are at least suitable for processor blocks where the boot software can be verified.

Keywords: Functional simulation, hardware verification, HW/SW co-simulation, digital electronics, logic simulator.

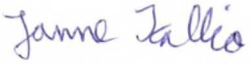
FOREWORDS

This Master's Thesis has been written at Nokia Networks IP Mobility Networks hardware development department in Espoo.

I would like to thank my supervisor Raimo Sepponen and my instructor Ari Hurtta for good comments and ideas. Special thanks go to Tapani von Rauner for his advice and checking the language. The whole staff of the hardware development department is thanked for their help.

Warmest thanks to my parents and my sister for their help during my studies. Finally I would like to thank my fiancée Hanna for the support that she has given me.

Espoo 28.5.2002



Janne Kallio

TABLE OF CONTENTS

SYMBOLS AND ACRONYMS

A2SU	AAL2 Switching Unit
AAL	ATM Adaptation Layer
AAL2	ATM Adaptation Layer type 2
API	Application Program Interface
ASM	Assembler
ATM	Asynchronous Transfer Mode
ASIC	Application Specific Integrated circuit
CA	Cabinet Address
CID	Channel Identifier
CPLD	Complex Programmable Device
CU	Computer Unit
CVE	Co-Verification Environment
DDR	Double Data Rate
DSP	Digital Signal Processing
DUT	Device Under Test
EDA	Electronic Design Automation
EEPROM	Electrically Erasable Programmable Read Only Memory
FDU	Floppy Disk Unit
FIFO	First In First Out
FPGA	Field Programmable Gate Array
HCE	Host Code Execution
HDL	Hardware Description Language
HW	Hardware
IP	Internet Protocol
ISS	Instruction Set Simulator
MB	Message Bus
MHz	Megahertz, 10^6 Hz
MXU	Multiplexer Unit
NEMU	Network Element Management Unit
NIU	Network Interface Unit
NIWU	Network Interworking Unit

ns	Nanosecond, 10^{-9} s
OMU	Operation and Maintenance Unit
PCI	Peripheral Component Interconnect
PIU	Plug-In Unit
ps	Picosecond, 10^{-12} s
PSP	Processor Support Package
PWB	Printed Wiring Board
RB	Repeater Bus
RNC	Radio Network Controller
RTL	Register Transfer Level
SDF	Standard Delay Format
SDRAM	Synchronous Dynamic Random Access Memory
SFU	Switching Fabric Unit
SoC	System on Chip
SOMA	Specification Of Memory Architecture
SPU	Signal Processing Unit
SRAM	Static Random Access Memory
SW	Software
TBU	Timing and Hardware Management Bus Unit
TDM	Time Division Multiplexing
us	Microsecond, 10^{-6} s
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WDU	Winchester Disk Drive Unit
Ω	Ohm

1 INTRODUCTION

Today as electronic products get more and more complex verification becomes more critical. Time to market is essential so there is no room for errors in design. One single error can severely delay a product and cause massive losses in the market. The sooner the errors are found, the lower the costs of correcting them will be. There are many different solutions to verify designs and simulation is one of them.

The 3G core network elements developed by Nokia consist of plug-in units, which all have their own function. These plug-in units are designed internally in Nokia and also partly plug-in units are designed in collaboration with partner companies. This Master's Thesis looks into verification of the digital design of the plug-in units using a logic simulator. Only the logical function is verified without any consideration to issues like signal integrity or EMC.

Verification is a process used to demonstrate the functional correctness of a design [1]. This process is becoming more important as electronic products develop. Today, in the era of multi-million gate ASICs, reusable Intellectual Property, and System on Chip (SoC) designs, verification consumes 70% of the design effort [1]. A plug-in unit consists of different components and of a printed wiring board. Thus it is easier to verify than a SoC or ASIC, but the time consumed in plug-in unit verification is growing all the time due to increased complexity and design size.

The functional correctness of a design can be verified before an actual prototype is finished by using simulation. In computer science the term simulation means reproduction of a real event with a computer based model [2]. In this Master's Thesis simulation means computer aided modelling of the function of a plug-in unit. Thus a functional model for every component and schematics of the design are needed.

A system is called an embedded system if it consists of an electronic device that performs a certain function and software that controls the device [3]. Most of the core network element plug-in units designed in Nokia are embedded systems. An embedded system can be verified by running software in a prototype plug-in unit but the verification cannot begin until prototypes are ready. To speed up the design process co-verification tools have been developed by the EDA industry. With these tools, hardware and software can be verified before the first prototypes are ready.

The main purpose of this Master's Thesis is to consider how plug-in unit simulations should be carried out. This will be evaluated with case studies that are done during the writing of this Master's Thesis. These case studies will also reveal to some extent which parts of the plug-in units can be simulated. Both hardware and hardware/software co-simulations are examined in this Master's Thesis. The purpose is to investigate how different simulation methods can be used in plug-in unit design process and what are the benefits to software and hardware development. It is not the purpose of this Master's Thesis to give exact instruction how to use simulation tools.

The Master's Thesis is arranged so that at first we look into the hardware and software development process and to the plug-in units that are designed. Some theory about verification is also studied. Chapter 3 introduces tools used in simulation, which include simulators and functional models for components. In the fourth chapter simulation is described. This includes hardware simulation, hardware/software co-simulation and testbench creation. Chapter 5 includes simulation case studies and the results from them fulfilling the purpose of this Master's Thesis. In the last chapter there is a summary and some further development ideas.

2 BACKGROUND

This chapter will at first introduce the electronics and software design processes. Both processes are used by programs to guarantee quality in HW and SW development. Programs are started when customers need new features to products or there are internal development needs. This latter can be for example that a component is not manufactured anymore and a plug-in unit must be re-designed. The result of a program is a new release of the product which usually includes new hardware and software. In this Master's Thesis the interesting development processes are the electronics and software design processes. The processes described here are used in the core network element development in Nokia Networks Finland. Core network elements and base stations form a mobile network. From now on, in this Master's Thesis, Nokia refers to the hardware development department in Finland, which is responsible for the core network element hardware. At the end of this chapter the IPA 2800 plug-in units and verification theory are described

2.1 Electronics Design Process

Electronics design process describes how a hardware product, here namely a plug-in unit, is implemented. The process consists of different activities, which are described here. This design process is described in more detail in reference [4]. Different process phases and milestones are shown in Figure 1.

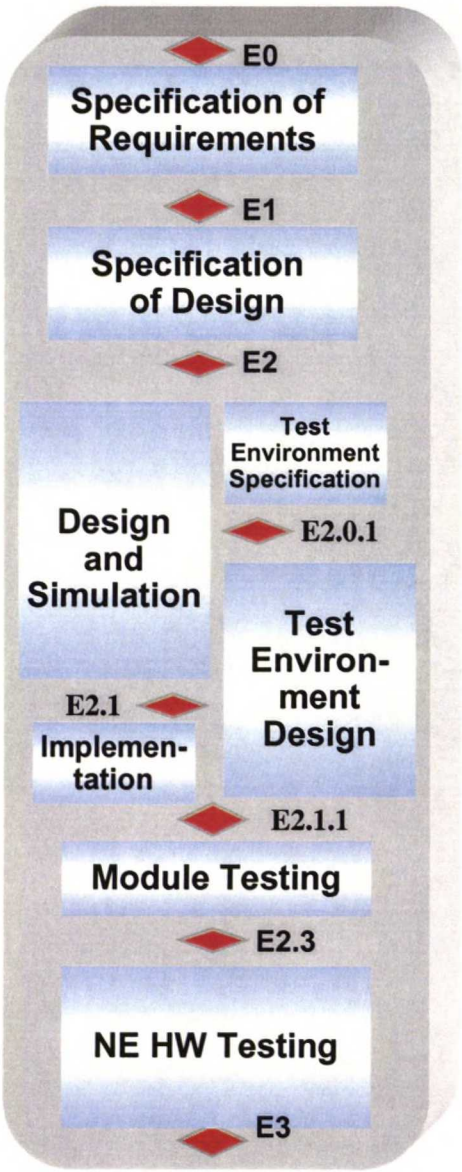


Figure 1 Electronics design process phases [4].

The design process starts from the specification of requirements. This includes the specification of operational environment, performance targets, the use and maintenance of the product, power consumption targets, reliability targets and interfaces. Also other requirements can be specified if necessary. The requirements are described in the Feature Requirement Specification, which is written and reviewed in this phase. Other activities are such as critical component recognition and co-operation with software development. The process will reach E1 milestone when the Feature Requirement Specification is ready and reviewed. The next phase will be the Specification of Design. Main activities are specification of the electronics design and test environment design. The test environment design can also be specified for a hardware project. If a plug-in unit consists of different blocks, implementation of blocks is also specified. This phase is finished when the plug-in unit implementation specification is ready and reviewed.

After milestone E2 there are two parallel paths, design path and test environment path as seen in Figure 1. The design path is described at first. Between the milestones E2 and E2.1 the main activities are block design, plug-in unit design, block and plug-in unit simulations. Plug-in unit and block circuit design is done and verified by simulation. Block and plug-in unit mechanical and thermal design is also done. EMC and testability issues are taken into account. The phase is finished after the plug-in unit design review is held. Implementation is the next phase in the design path. Main activities are printed wiring board (PWB) design and module test plans. High Speed simulations are done also in this phase. After the PWB design is ready, the first prototypes are ordered. Such activities as Software design and ASIC/FPGA testing are supported. Module test plan is written to the plug-in unit and also to different blocks. This phase is finished once the prototypes are ready.

The test environment path starts from milestone E2 and the first phase is the Test Environment Specification. Main activities are the specification of test structures needed for electronics module, electromechanics module, hardware system and network element hardware system. This includes for example test cartridges, operating system software and device driver software. Plug-in unit designer must ensure that these test structures can be used to module test the plug-in unit. E2.0.1 milestone is reached when all test structure specifications are written and reviewed. The next phase in test environment path is test environment design. Structures needed for the plug-in unit module testing are designed and tested as far as it is possible without the actual prototypes. Phase is finished when the test environment for the module testing of prototype plug-in units is ready.

Plug-in unit and block module testing is done during the next phase. As the plug-in unit module testing progresses and errors in design are found. These errors are corrected using blue-wire fixes or, if more drastic fixes become necessary, new design iterations are done. This ensures the correct functionality and manufacturability of the plug-in unit. Software developers use the plug-in unit to module test the plug-in unit's software. Co-operation with software and hardware people is necessary during this phase. E2.3 milestone is reached when the plug-in unit has been released for network element hardware testing and internal production in the plug-in unit implementation review.

Network element hardware testing is the last phase in this design process. In this phase the plug-in unit is tested as a part of a network element. Milestone E3 is reached when the plug-in unit final review is held. In this review plug-in unit documentation will be approved. With this approval plug-in unit and blocks are released for general use. The design process is finished and plug-in unit will enter the maintenance phase of its life cycle.

Every plug-in unit has a designer, who is responsible for the whole plug-in unit. Different blocks have their own designers, who are responsible for their blocks. Development tasks have been divided to different designers, because plug-in units are so complex. Blocks are designed so that they can be used in different plug-in units. Thus all blocks are not necessarily designed during the plug-in unit design process. Maintenance of the plug-in unit is also the responsibility of the plug-in unit's designer.

2.2 Software Design Process

Since most of the plug-in units are embedded systems, there is software that controls the hardware. From the hardware point of view the software boot, device drivers and operating system are the most interesting ones. Application software runs on top of the embedded software. The purpose of the embedded software is to provide a platform, which is not dependent on the hardware where the application software is executed. This means that a change in the hardware should only change the embedded software not the application software. Here the software development process will describe how the low-level software is developed.

The software development process is a separate process and starts later than the electronics design process. Plug-in unit specification of requirements is developed in co-operation with the hardware and software designers. The same is done to the plug-in unit implementation specification and after it is finished the software development can begin. This means the implementation of boot, device drivers and modifications to operating system if necessary. The software development process tries to map to the hardware development so that the software is ready when it is needed. There is no co-development process during the design phase where the two processes would advance in parallel. [5]

Software developers start developing the software boot, which is used to initialise the hardware. The boot is written according to the hardware implementation specification and verified when the prototypes arrive. During the boot development also the device drivers are developed. The software developers study device data sheets and implement device drivers for the operating system in use. These are also verified using the prototypes.

When the first prototypes arrive, the hardware designers check that there are no major electrical problems in the plug-in unit and then they are sent to the software designers. The debugging of the boot begins and problems are solved together with hardware and software designers. After the boot and the operating system are verified, the verification of the device drivers can start. Device drivers are tested and problems are corrected in hardware and software when found. Usually the problems are easier to correct in software than hardware.

The software verification and development can continue after the E3 milestone is achieved in electronics design process. This is due to the fact that the hardware can be verified to be correct before all tests to the software are completed. There can be also problems in integrating application software on top of the embedded software. These problems can usually be corrected in embedded software. This means that the embedded software verification and development continues after the electronics design process is completed.

2.3 IPA 2800 Platform Plug-In Units

IPA 2800 is Nokia's platform for the third generation mobile core networks. Network elements are developed using this platform, for example radio network controller (RNC). A network element consists of many different plug-in units that perform different functions. These plug-in units can be divided into four categories: ATM switching and multiplexing, control and computing, network interfaces and support functions [6].

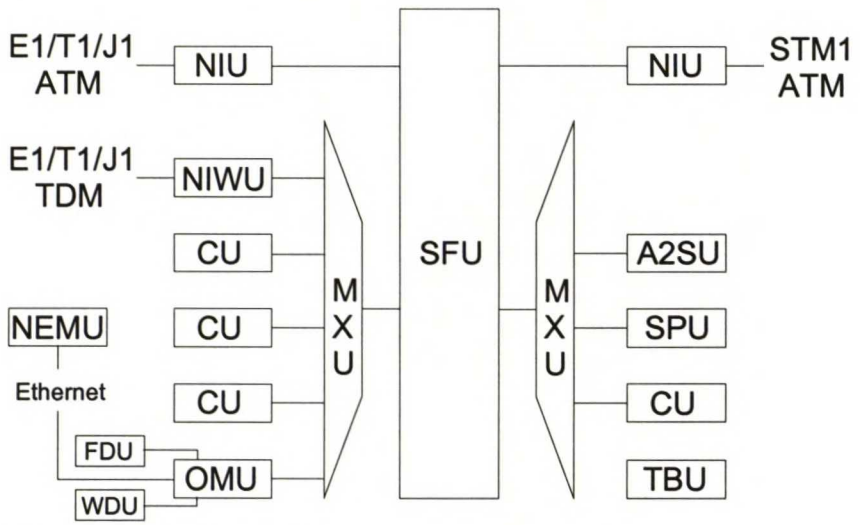


Figure 2 Generic block diagram of IPA 2800.

Generic block diagram of IPA2800 is shown in Figure 2. The blocks in Figure 2 are plug-in units and connections to other network elements are just described as examples of possible connection types. The number of plug-in units varies in network elements and there is a spare plug-in unit for one or more active plug-in units to achieve higher reliability of the system. Naming of the functional units is done using generic terms to describe the function of the unit. These terms will be explained later in more detail.

ATM switching fabric (SFU) and multiplexing (MXU) units handle the multiplexing and switching functions. The switching fabric unit consists of a switch fabric, input and output ports and a unit computer. The purpose of the unit computer is to control the unit. Switching fabric unit can be connected directly to other units without the multiplexing unit. The inputs to multiplexing unit have lower bit rate than the inputs of the switching fabric unit. The purpose of the multiplexing unit is to connect lower bit rate units to switching fabric. Main hardware blocks in the multiplexing unit are the unit computer, connections to switching fabric and tributary units and ATM layer processing. The multiplexing unit handles ATM layer processing and the switching unit only routes the packets to right output ports.

Control and computing units are in Figure 2 control computer unit (CU), operation and maintenance unit (OMU), network element management unit (NEMU), AAL2 switching unit (A2SU), Winchester disk drive unit (WDU), floppy disk drive unit (FDU) and

signal processing unit (SPU). Control computer unit and operation and maintenance unit have both the same hardware and the function is determined by software. The difference in these plug-in units is that OMU is a master computer and its purpose is to perform centralised parts of the system like controlling hard disk (WDU) and magneto optical disk drive (FDU). WDU and FDU are simple plug-in units, which have hard disk or magneto optical disk drive connected to the circuit board. Control computers (CU) are used in different functions depending on the software they run. These functions include controlling the system, protocol processing, management and maintenance tasks. Network element management system unit (NEMU) is connected to the system with Ethernet. The plug-in unit is a computer unit which provides graphical user interface and is responsible for network element maintenance tasks. NEMU can be used locally by connecting a monitor and keyboard to the plug-in unit or it can be accessed via Ethernet network. Signal processing unit provides support for e.g. data compression and ciphering. The unit has four unit computers and four daughter cards, which each have 8 digital signal processors and some memory. SPU provides a general-purpose platform, which can be used in the above mentioned functions. AAL2 switching unit (A2SU) is the same unit as SPU without the daughter cards. This unit performs switching of AAL type 2 packets i.e. the unit takes virtual channel with packets containing different CID as input and switches the packets to new virtual channels depending on the CID of the packet.

Network interfaces provide connections to different networks. These are divided into two categories: network interface units, which are used for connecting network element to various types of transmission systems, and network interworking units, which are used for connecting the network element to non-ATM transmission system e.g. TDM E1. Both NIU and NIWU have a unit computer that controls the unit. They can be connected to switching fabric directly or to multiplexing unit. If the plug-in unit is connected directly to switching fabric, it has an ATM Layer processing block. Other main hardware blocks are the physical connection block and data conversion block, which provide for example IP over ATM functionality. Connections to other transmission systems can be electrical or optical.

Last of the four categories is support functions. As the name already indicates these plug-in units are used for support functions and the functions are timing and synchronisation and power feed. The timing and synchronisation is divided into two plug-in units. The first plug-in unit handles synchronisation by receiving a signal from upper network element and delivers synchronised timing as output. The second plug-in unit is timing buffer which distributes the timing received from timing and synchronisation plug-in unit to other plug-in units e.g. NIU, CU. These plug-in units have also the hardware management system power feed and bridge node, which will be described later in more detail. Last of the support functions is the power feed plug-in unit, which distributes power to other plug-in units. The unit controls also the fans in the network element and has a connection to a hardware management system's master node.

The hardware management system provides a fault tolerant message passing between operation and maintenance computer and other plug-in units. Every plug-in unit has a connection to the hardware management system and it is called a hardware management

system slave node. This system is used, for example for changing plug-in unit operation state or even shutting down the plug-in unit's power feed. As mentioned earlier, the hardware management system has its own power feed. Thus it is not dependent on plug-in unit power feed. The hardware management system slave nodes connect to a bus that is connected to a bridge node. The Bridge node is connected to a management node which is in the operation and maintenance computer and is used for controlling the bus. The system is divided into subsystems which are connected to each other through bridge nodes.

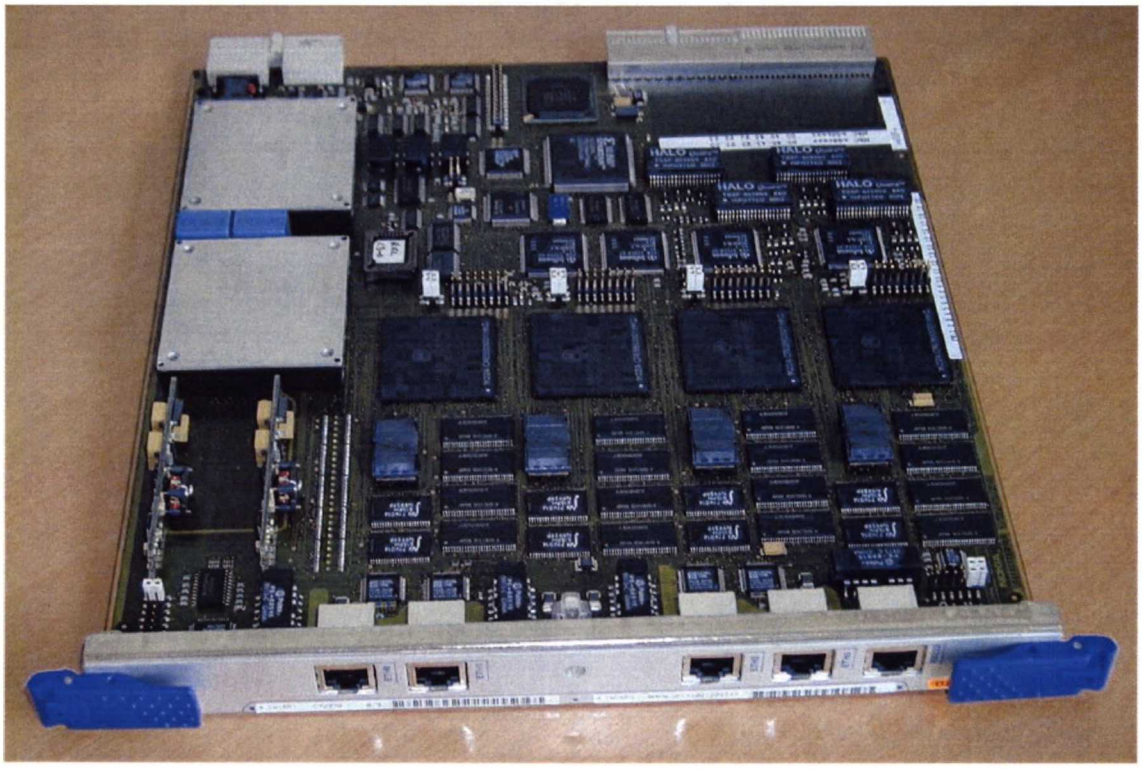


Figure 3 Interworking plug-in unit (265 mm x 285 mm).

Interworking plug-in unit is described in Figure 3. IPA 2800 plug-in units all have the same mechanics. Their size can vary depending on the space which the components need. There are plug-in units which are twice as thick as the plug-in unit described in Figure 3. Timing and Synchronisation plug-in units are half the width of a normal plug-in unit. Plug-in units are connected to a cabinet, which has four subracks where plug-in units are connected. One subrack can be equipped with maximum of 20 plug-in units. There are fixed slots for plug-in units like power feed plug-in unit and general-purpose slots where different plug-in units can be equipped.

2.4 Verification Theory

Verification of a product takes a lot of time in the whole design process. The theory presented here is based on System on Chip designs. Most of the plug-in units are System on Board designs which means a system that has processor, memory and peripherals

connected to a PWB. The principal of verification is the same whether the product is a System on Chip or a System on Board. Of course, the System on Board is easier to verify since a logic analyser can be used to check a value in an internal bus. Components in System on Board design are mainly discrete. Thus you should be able to trust that they function correctly.

Hardware designers design products according to specification. In a System on Board design, the designer decides which components fulfill the requirements in the specification. The designer has to interpret the data sheets of the components to be able to draw the schematics. As mentioned earlier in electronics design process, reviews are held to ensure that the designer has designed everything correctly. Often this is not sufficient, therefore errors in design are usually found. Simulation can be used to minimise the risk of errors in design.

Reconvergence model is a conceptual representation of the verification process. It is used to illustrate what is being verified. The design process can be thought as a set of transformations. In HDL model design these can be, for example RTL coding from specification and synthesising RTL code into a gate-level netlist [1]. These same transformations are done in board design. The specification is transformed into schematics and the schematics are transformed into PWB to give a few examples. We can see in Figure 4 that the verification is a second reconvergent path. This is in System on Board design for example the phase from specification into schematics. This means that the simulation results must be checked against the specification to really verify if everything is designed correctly.

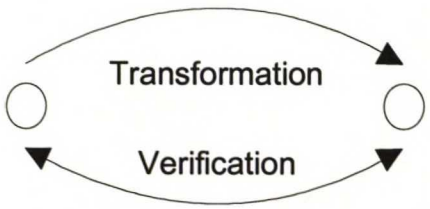


Figure 4 Reconvergent paths in verification [1].

Verification should also reduce the human factor of errors. The designer or design team interprets the specification and design the product using the interpretation of the specification. This is not as difficult in System on Board design as in System on Chip designs, but there are cases where the wrong interpretation leads to problems. Usually the function of discrete components is found from data sheets. When one reads these data sheets, one makes an interpretation of the function. This is described in Figure 5 using HDL coding as an example. The designer reads the specification and interprets it. Then he verifies the RTL coding against his interpretation of the specification. In System on Board designs the designer has understood a data sheet of a component in some way and then the designer verifies the schematics with the interpretation of the function. Thus if the interpretation is wrong, the verification does not remove the error.

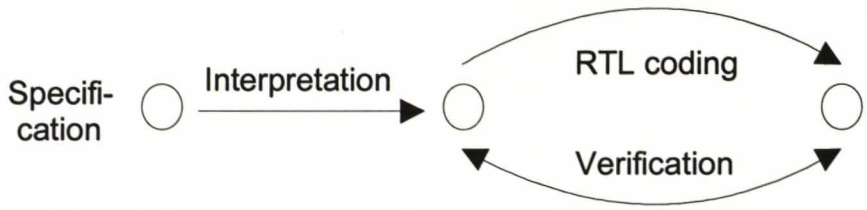


Figure 5 Reconvergent paths in ambiguous situation [1].

There are different ways to eliminate the human-introduced error, for example automation and redundancy. Automation takes the human intervention in a process away but it is not useful in processes that are not well defined like hardware design. Redundancy is another way to reduce the human-introduced errors. Figure 6 describes the situation in HDL coding. The redundancy is done by using different persons as designer and verifier. They both read the specification and make their own interpretation. This way it is more likely that the design is correct and both have not misinterpreted the specifications in the same way. In System on Board designs the same situation could come up in component specification. The designer would read the specification and draw the schematics according to that. The verifier will then read the same specification and verify the design against that.

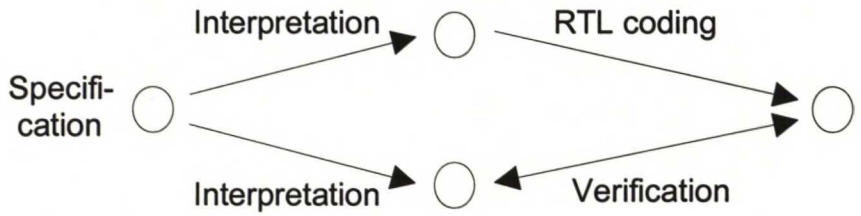


Figure 6 Redundancy in an ambiguous situation [1].

The term verification can be confused with the term testing. Testing is done to ensure that the product was manufactured correctly and verification is used to verify that the design meets its requirements.[1] Testing vs. Verification is explained in Figure 7 using HDL coding as an example. HW design phase includes all the phases necessary to get the netlist ready for the silicon vendor. The designed netlist is verified to be correct comparing it to the specification. When the chip is manufactured, it is tested to be correct using the netlist. In System on Board design equivalent situation would be the prototype manufacturing from the schematics. This would include the PWB manufacturing which would also be tested. Schematics of the design are verified to be correct and the prototype is tested to be correctly manufactured using the schematics that are given.

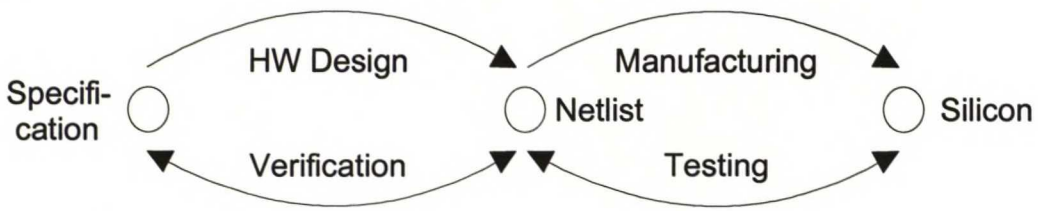


Figure 7 Testing vs. verification [1].

Testbench is used to verify the functionality of a design. The purpose of the testbench is to create stimulus, apply stimulus to the device under test and check the result. Stimulus is the data that is assigned to the input ports of a design i.e. the tests done to the design. The output ports are checked and determined whether the respond to the stimulus was correct. Testbenches can be done with HDL or specific hardware verification languages. [7]

Verification is problematic because the designs are very complex. The testbench should test all the possible combinations. ASICs and FPGAs are designed with HDL and testbenches are usually also written with HDL, which is designed for implementation of integrated circuits. Thus it does not have special features that would be helpful in testbench creation. VERA and e are examples of languages that are designed specifically to verify hardware products. These languages provide high-level data structures available only in object-oriented languages such as C++ and Java. They also provide randomisation capabilities which can be used to automate the stimulus. In addition to high-level data structures, these languages provide constructs necessary to model hardware concepts like buses and concurrency.[7]

Functional verification flow of an ASIC can be divided into three major phases: strategy phase, testbench phase and regression & coverage. Strategy phase includes the identification of the test cases, how these are done and verified. Testbench and creation of the test cases is done in the testbench phase. Once all the cases are passed, the verification moves to the regression and coverage phase. The tests are continued in this phase until the coverage of the test cases is adequate. Regression tests are done after this. [7] The verification flow described here is one possible flow and it can be modified to be suitable for the design in verification. System on Board design flow has basically the same steps as the ASIC verification flow. Coverage is not determined the same way as in ASIC design but the flow is still suitable for System on Board designs.

3 TOOLS

This chapter will introduce the tools used in simulations. At first the simulators and the HW/SW co-verification environment is introduced. Next different models will be described and at the end of this chapter there is a short description of schematic to HDL conversion tool.

3.1 Simulators

There are many different hardware simulators available. In this Master's Thesis digital designs are simulated using a logic simulator. This chapter will give an overview on simulators and concentrate more closely on simulators used in Nokia. These simulators can be divided into two groups: HDL and schematic based simulators.

3.1.1 HDL Simulators

HDL simulators are used to simulate designs written with hardware description languages. Some simulators are designed to be used with Verilog or VHDL and some can simulate mixed language designs. HDL simulators are mainly used in FPGA and ASIC designs. Many EDA vendors have their own HDL simulators for example Mentor Graphics has ModelSim, Synopsys has Scirocco and Cadence Design System has NC-Sim. ModelSim is a mixed language simulator. Thus it supports both VHDL and Verilog. Scirocco supports only VHDL. NC-Sim is also a mixed language simulator. There are many HDL simulators available provided by these EDA vendors and others. The ones mentioned here are market leaders in some area. ModelSim is the market leader in mixed simulations and Cadence's pure VHDL simulator is the leader in VHDL simulation [8].

ModelSim simulator is used in this Master's Thesis and its features are presented below. The VHDL or Verilog code is first compiled into a design library with the tools that ModelSim provides. The compilation checks the semantics of the code and after that the simulation is executable. When the simulator is started, design elaboration is done. This means going through the design hierarchy and checking that everything is connected properly. Execution is the next phase which means running the simulation for a predefined amount of time.

ModelSim provides the basic features like most electronic simulators. User interface of the tool is graphical and there are different windows for providing information of the design like waveform window. The source code can be viewed during the simulation and the code can also be run by single stepping or by setting brake points. Debugging hardware description language is not very clear because HDL code is executed in parallel. This means that many processes are executed without advancing the simulation

time. Resolution of the simulator is 1 ps, which means that the simulation can be advanced one 1 ps at a time. The resolution can be changed to for example 1 ns. Signal values can be forced during the simulation to a specific state. The values which can be used are the same as the ones defined in the HDL language. Forced signal can be frozen to a specific state, which means that it cannot be changed by the simulation models in the design. Other possibility is to drive a signal to a desired state. This enables the simulation models to override the condition. In the design this would be equal to a pull-up or pull-down resistor. There is also a specific force command for clocks which is used by giving the duty cycle and pulse width.

There are interfaces which allow the simulator to be connected to other programs. In VHDL this interface is called Foreign Language Interface. This allows for example testbenches designed with C to connect to VHDL designs. Verilog has a similar interface which is called Programming Language Interface. This interface is also included in ModelSim. Simulation models can be connected with the SWIFT interface which is described later in more detail. ModelSim runs on various platforms including Windows, Sun Solaris, HP-UX and Linux. [9] Simulations at Nokia are mainly done in workstations using Unix or Linux operating system.

3.1.2 Schematic Based Simulator

Schematic based simulators use schematics of a design to simulate connections between components. To be able to use the simulator the schematics must be compatible with the simulator in use. This compatibility is usually not a problem when both tools are from the same vendor. The schematic based simulator used in Nokia is QuickSim Pro developed by Mentor Graphics. This tool is described below.

QuickSim Pro tool can simulate models, which are connected to the schematics through for example the SWIFT interface, and it supports simulation models written with HDL. The tool is actually a combination of two separate tools QuickSim II and ModelSim. QuickSim II simulates the schematic part and models connected directly under component symbols through supported interfaces. ModelSim simulates models which are written or the connection to the model is written with HDL. The resolution of QuickSim Pro is the same as in ModelSim. This must be set to be equal in both simulators separately.

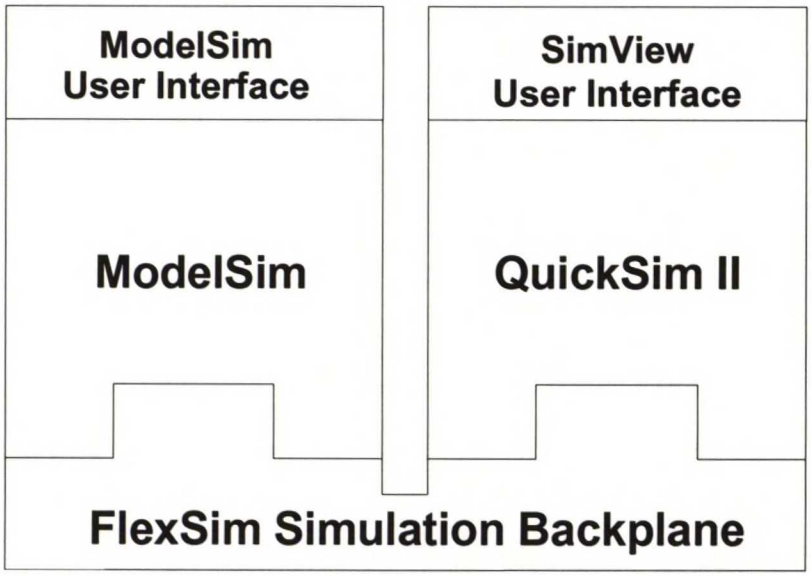


Figure 8 QuickSim Pro simulator architecture [10].

The architecture is described in Figure 8 which consist of the two tools and the FlexSim Simulation Backplane. When the simulator is started, it evaluates the design and partitions the design to both simulators. In case there are no HDL simulation, only QuickSim II simulator will be invoked. The FlexSim Backplane controls communication and synchronisation between the simulators. The simulators advance time independently and exchange data when necessary. Nets which are common to both simulators are called boundary nets. Events happening on boundary nets are communicated through the FlexSim Backplane. If there is no activity on boundary nets, no communication is needed.

QuickSim Pro user interface actually is two separate user interfaces which are the ModelSim user interface and SimView user interface for QuickSim II. The simulation can be run from both simulators. All features of ModelSim can be used in the simulation, but these are limited only to HDL parts of the design. QuickSim II has a graphical user interface and it has similar features like ModelSim. Signals can be selected to the waveform window directly from the schematics. Signals can be forced to a value from both simulators but these affect only the signals which are controlled by the simulator where the command was given.

Simulations done with QuickSim Pro are slower than simulations done in either of the two simulators separately [10]. This is due to the communication done through the FlexSim Backplane. The designs should be partitioned when possible so that the communication between the two simulators would be reduced to minimum. Stimulus can be created in either simulator. In ModelSim this would be written with HDL and in QuickSim II force files can be used.

3.2 HW/SW Co-Verification Environment

Co-Verification Environment is a tool which connects software debugger to hardware simulator. This environment allows the execution of software on virtual hardware which is simulated in a logic hardware simulator. Seamless is the co-verification environment developed by Mentor Graphics. It is the market leader in this area with 90 % market share in 2001 [8]. Features of the tool are described below.

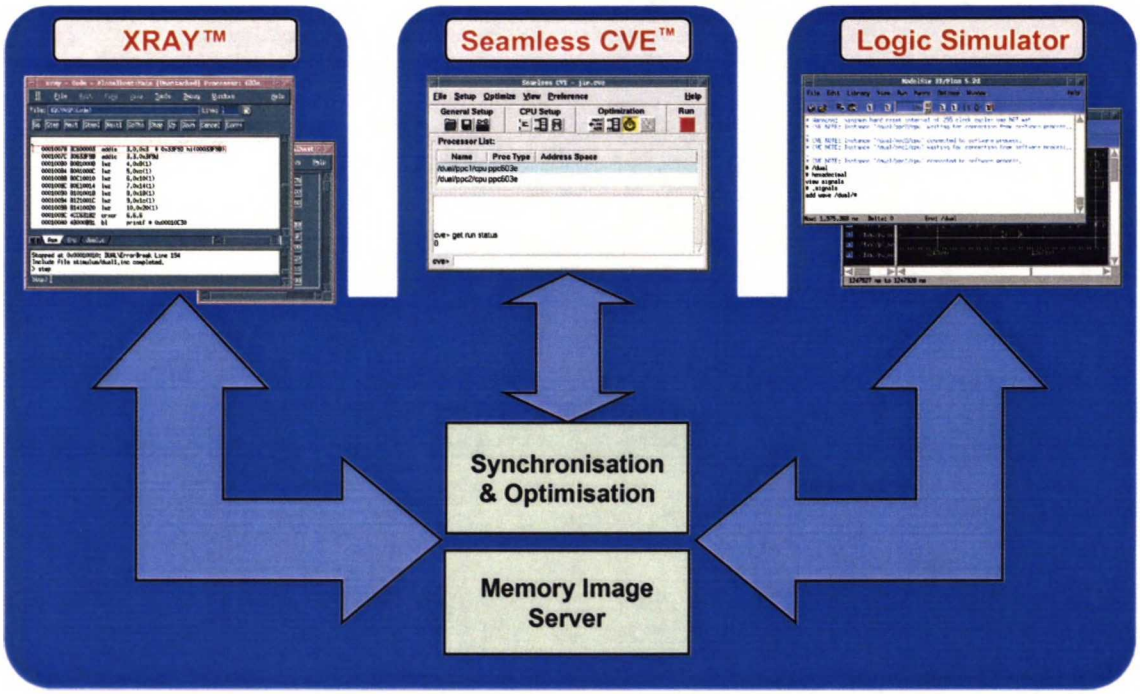


Figure 9 Seamless co-verification environment [11].

Seamless CVE is the connection between the Instruction Set Simulator (ISS) and the logic hardware simulator as shown in Figure 9. Synchronisation of the two simulators is handled by Seamless CVE and the two simulators communicate through the interface provided by Seamless CVE. Software is run on the ISS and the commands are transferred to the processor model, which runs the corresponding bus cycle in the logic hardware simulator. Simulation is controlled from the software debugger or from the logic hardware simulator. The simulation can for example be run so that the logic hardware simulator is set to run and the code is single stepped through the software debugger. The logic hardware simulator used in these simulations is ModelSim and the hardware simulation does not need any modification from pure hardware simulations except that special processor models are needed in the HW/SW co-simulation. QuickSim Pro can also be used as the hardware simulator, but as described in the previous chapter, the communication through the FlexSim Backplane makes the simulator run slower than ModelSim would run without QuickSim II. Processor models are the key element in HW/SW co-simulations since the software is run on processor. Mentor Graphics provides special processor models to be used in Seamless CVE (see chapter 3.3.4). To be able to use all features provided by Seamless CVE also the

memory models should be replaced with Denali memory models, which are described later in chapter 3.3.3. Seamless provides also generic memory models, which also support optimisations. These models include for example SRAM memory. The memory model is created by configuring the VHDL file provided with Seamless. Some glue logic may have to be created if the memory does not exactly match the generic memory model.

An ISS can run up to 100 000 instructions per second, but the logic hardware simulator is only able to run 1 instruction per second [12]. This means that the speed of the simulation is determined by the slow logic hardware simulator and the use of optimisations is necessary. Processors usually execute code from an external memory. There is usually an EEPROM or FLASH memory where the software boot is stored. In simulation, memory models contain the same data as in real product. When the simulation is started, the processor starts to read the code that is executed from memory. The code can be viewed from the software debugger but the actual data is retrieved from the logic simulator. This means that the ISS gives a command to the processor model to retrieve data from a specific address. Then the processor model in the logic simulator runs a bus cycle to the memory model. The memory model corresponds to the bus cycle and sends the data through the data bus to the processor model. Software debugger knows the data that should be retrieved with the logic simulator. If map check option is enabled from Seamless CVE, then Seamless will give an error message if the data retrieved using the logic simulator does not match with the data in the software debugger.

A read or write operation to a memory can be verified to be correct when it has been done several times. Memory connection can be also checked using a walking 1's algorithm, which goes through the address and data pins by setting one pin to logical 1 at a time. This algorithm is created using XRAY macros and it can be received from Mentor Graphics. After the memory connection is verified there is no point in verifying it again. Denali memory models use the Memory Image Server which can be accessed by the ISS directly. This way the slow logic hardware simulator does not have to perform memory write or read operations and the data can be retrieved from the Memory Image Server directly by the ISS. This feature is one of the Seamless CVE optimisations. By using the memory optimisations, the performance of the simulation increases significantly. ISS can run faster when the logic hardware simulator does not need to perform bus cycles to optimised memories.

Seamless CVE controls the processor memory space. The actual memory devices in the design need to be mapped to processor memory space. Memory optimisations can be turned on or off during the simulation and the optimisation can be applied to one address range or several. For example the FLASH memory could be optimised to be able to run the simulation fast to the point were the writing to SDRAM starts. When this starts, the SDRAM access can be verified from the logic hardware simulator. This allows using the optimisations when necessary or disabling them when a detailed information is needed. Time optimisation is another feature that allows the simulation to run faster. This optimisation is used with the memory optimisation and it allows the logic hardware simulator to stop completely when there are only memory accesses. The logic hardware

simulator starts running again when there is an access to external hardware device. This means that the software is run ahead of the hardware. If there are timing loops where the software expects the hardware to be running, the time optimisations cannot be used. An example of this would be a software reset which would be followed with a delay loop. If time optimisations are turned on, the logic simulator does not advance during the delay loop. This can mean that the hardware does not have time to recover from the reset since the logic simulator is not advanced.

The software debugger comes with the processor model. There can be several debuggers available, but usually Mentor Graphics XRAY debugger is included in the model. Memory contents can be viewed and edited with the XRAY debugger. The tool allows also viewing the registers in the processor model that the software uses. Memory contents can be loaded or dumped through Seamless CVE user interface. Executable software is loaded with the software debugger and stored into the memory image server. The memory contents can also be updated by loading a memory image file from the Seamless CVE user interface. Software needs no modifications to run on Seamless CVE since the processor models the instruction set of a real processor and the memory devices are also modelled to function like the real memory device. The limitation of the system is basically the speed at which it runs. This means that very large software packages can cause problems in simulation or the simulation takes too long to give any value to designers.

3.3 Models

In simulation one must have models to components since without models the simulation would not be possible. At least the key components should have models and, of course, best would be to have models for every component. Models can be written with hardware description language or with software languages like C. The main requirement in deciding the modelling language is that models can be used in different simulators.

3.3.1 HDL Models

There are two main hardware description languages VHDL and Verilog. VHDL is widely used in Europe and Verilog in the USA. VHDL was developed by the U.S. Department of Defence sponsored projects. It was intended to be used to document digital circuits without any technology constraints. This would allow the Department of Defence to replace an obsolete component without huge amount of redesign.[13] Now VHDL is used to design CPLDs, ASICs and FPGAs. VHDL was adopted as an IEEE standard 1987 [14].

Verilog was originally developed by Gateway Design Automation as a proprietary language. This happened around 1984. At this time, the language was not standardised and it modified itself almost in every revision that came out within 1984 to 1990. It was noticed that the language would not survive as a proprietary language. Therefore

Cadence, which bought Gateway Design Automation in the late 1990, opened the language to other vendors. In 1995 Verilog became an IEEE standard. [15]

VHDL is a strongly typed language and it supports hierarchical design. Entity is one basic VHDL element. An entity describes the pins of a component. The internal function of an entity is called architecture. This describes the inner function of a component i.e. the response to input stimulus. An entity and architecture pair can be used as a component. Designs can have several components which are introduced and instantiated in the architecture. This means that you can write a VHDL model of a component once and use it several times in your design instead of writing the same code several times. In simulation there is usually a top-level which entity has no ports. This level is the testbench and it is used only to connect the design under test to the input stimulus. Component instantiation means that the port of a component is connected to a signal, which is then connected to another component or used as an internal signal. This is very similar to drawing schematics where the signals are nets which are connected to components.

Models used in plug-in unit simulations can be written with VHDL or Verilog. This basically is determined by the component vendor's choose of language. HDL models usually model FPGAs, CPLDs or ASICs. These models can have different levels of abstraction. FPGA/CPLD design flow, which is described in Figure 10, shows the different phases and the different VHDL abstraction levels. The design flow starts by writing the RTL VHDL code. In FPGA design flow this is also used to verify that the function of the circuit is correct. Next phase is synthesis and pre layout simulations which are done using gate level VHDL. After place and route the post layout simulations are done with the timing information included in VHDL netlist.

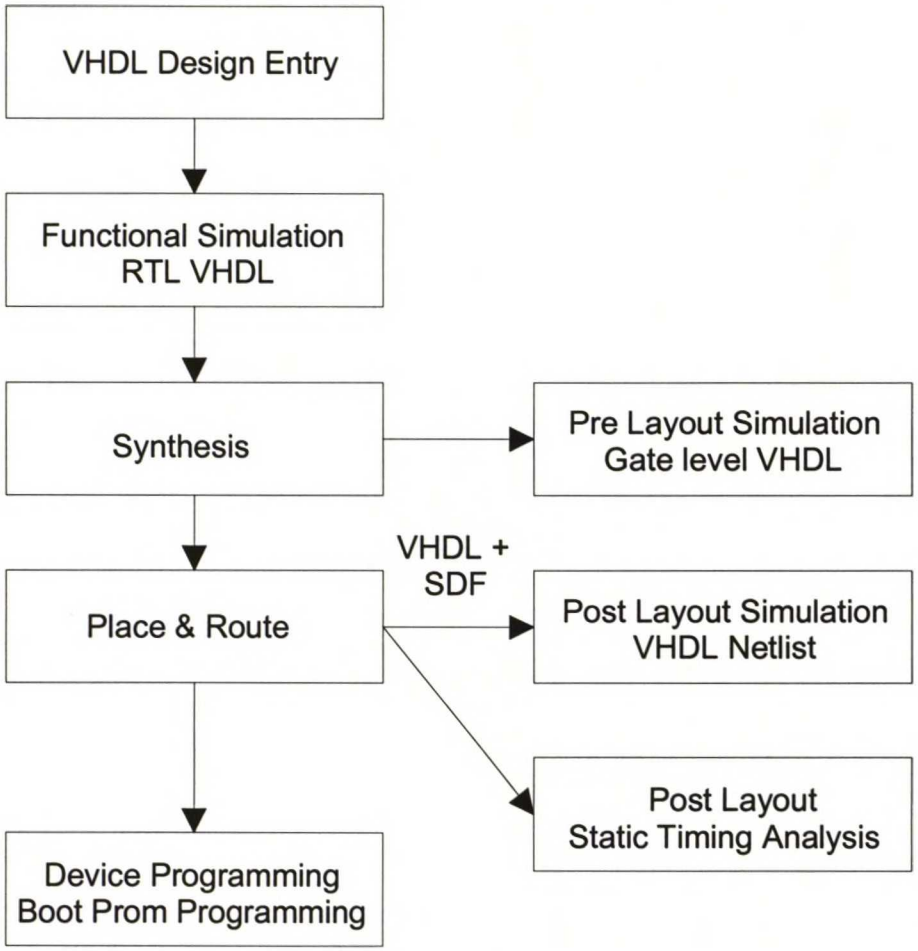


Figure 10 FPGA/CPLD design flow and file formats [16].

ASIC design flow does not differ much from the FPGA/CPLD design flow when the interest is in different VHDL abstraction levels. The different abstraction shown in Figure 10 can be used in plug-in unit simulation. Of course, if the FPGA, CPLD or ASIC is developed by a component vendor, all of these levels may not be in use. It may also be impossible to use VHDL netlist with timing information since the simulation speed decreases when the abstraction level is lower. The fastest one to simulate is RTL VHDL and it can be used almost in every case. If the designed circuit is not that large, the VHDL netlist with timing information does not reduce the simulation speed to the level where it is impossible to get any results. This may help in plug-in unit simulations where the timing is critical and needs to be verified.

Circuits developed in Nokia are designed with VHDL. The process described in Figure 10 applies also to Verilog designs. ModelSim can simulate mixed language models. This way the models received from component vendors can also be designed with Verilog. These models have the same abstraction levels, but simulation speed with the timing information included is faster than in VHDL designs [16]. Verilog models can be connected to VHDL testbenches with some limitations, for example the ports in Verilog design must be named [17].

Components, which have no models, can be modelled with HDL. In Nokia these are done with VHDL. Simple VHDL models can be written for example to pull-up or pull-down resistors. This model is very easily written with only a few lines of code. Other example of a model, which function can be modelled with VHDL, is a differential line driver and receiver.

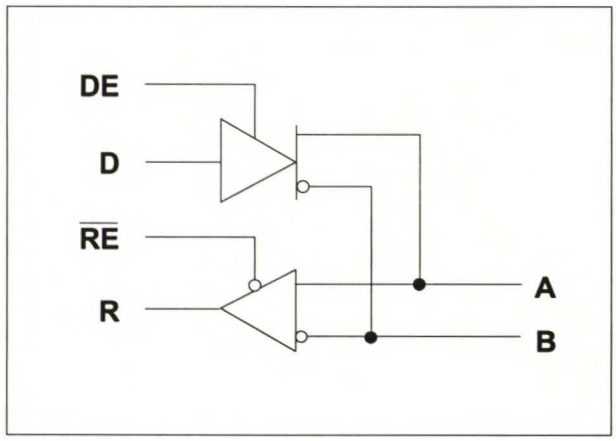


Figure 11 Line driver and receiver logic diagram [18].

The logic diagram in Figure 11 describes the function of a line driver and receiver. There are three input ports, one output port and two input/output ports. The logical function of the differential signals can be modelled with a combination of logical one and zero. If the signal sent to port D is zero, then line driver will drive zero to port A and one to port B. The receiver sees this as a zero and drives zero to port R if the enable signal is asserted. This kind of model can be used to check the schematic connection and functional correctness of a design. A line driver and receiver model described above is only about ten lines of VHDL code so it is fairly easy to implement.

3.3.2 SmartModels

SmartModels are developed by Synopsys incorporated. The SmartModel Library is a collection of over 3000 binary behavioural models of standard integrated circuits supporting more than 12000 different devices. The library includes microprocessors, controllers, peripherals, memories and general-purpose logic. SmartModels connect to hardware simulators with the SWIFT interface which is integrated with over 30 commercial simulators. [19] Depending on the simulator used, SmartModels can be run on various platforms including Windows NT, Linux, HP-UX and Sun Solaris [20].

SmartModels represent integrated circuits and system buses, like the PCI bus, as "black boxes" instead of modelling them at gate level. This means that the model accepts input stimulus and responds with appropriate output behaviour. Such models provide improved performance over gate level models and at the same time they protect the proprietary design created by the semiconductor vendor. [19]

There are two basic types of SmartModels full-functional models and bus-functional models. Full-functional models simulate the complete range of device behaviour. Most SmartModels are full-functional models. Bus-functional models simulate all device cycles and also two types of bus-functional models exist. The first is hardware verification models which are controlled by processor control language (PCL), a language that is similar to C. The second type is FlexModels which are controlled with Verilog, VHDL, C or VERA testbench. [19]

SWIFT is a standard EDA event-level simulation interface developed by Synopsys. SWIFT interface enables multiple simulators to use models from the same SmartModel library. Each simulator provides a standard model interface specified by SWIFT that allows it to load the same SmartModel library. When the simulator encounters a SmartModel during simulation, it uses a set of SWIFT functions to create and configure the model, to map to its ports and to set its time units.[21]

SmartModels are configured using SWIFT parameters when they are instantiated into the design. These parameters include an instance name, timing version and delay range. In some models memory image file or other command file is also required. These parameters are given for example in VHDL as generics. FlexModels have basically the same parameters but the instance name must be unique to be able to use several instances of the same model in one simulation. [21]

There are common features in most of the models. All SmartModels use 64-bits to compute elapsed simulation time. If this is exceeded, the models behave unpredictably. Models can be reset to their nominal state any time during the simulation. The model-reset operation resets the internal state variables but not the input port values, attribute values or mode settings. SmartModels use a logic value system based on the IEEE 1164.1 VHDL, nine state, multivalue logic system shown in table 1.[19]

Table 1 SmartModel logic values [19].

Symbol	Meaning
0	Strong 0
1	Strong 1
X	Strong X
L	Weak 0
H	Weak 1
W	Weak unknown
Z	High-Impedance
U	Uninitialised (treated as unknown)
D	Do not care (treated as unknown)

All SmartModels have at least one timing version available in the library. Models can check automatically if timing constraints are violated. This feature can be enabled with SWIFT parameters. If a timing violation occurs, the model writes an error message to the simulator control window. Timing checks include for example component specific

set-up time and when the component vendor has specified, nominal or worst case timing values can be used. It is also possible to create own timing information to a specific component.

Almost all basic logic integrated circuits are modelled in SmartModel library. An example of these are AND gates, multiplexers and buffers. Synopsys provides symbols of the models which can be connected to a custom symbol in schematics. The pins on the model symbol must be connected to the custom symbol and after this the models can be used without any modifications in QuickSim Pro simulations. The models can be used also in VHDL based simulations. ModelSim provides a program which automatically generates a VHDL wrapper to a specific model. This wrapper can be connected to VHDL with component instantiation.

FlexModels are models that represent the bus functionality of microprocessors, cores, digital signal processors and bus interfaces. They are essentially advanced SmartModels. These models have cycle-accurate core and built in timing shell, so they can be run in function only mode or with timing information included.[22] FlexModels can be controlled with different methods depending on the simulator used. When using QuickSim II, the only available control method is C. VHDL and Verilog control is supported in simulators that support VHDL and Verilog. VERA can also be used with ModelSim to control FlexModels.

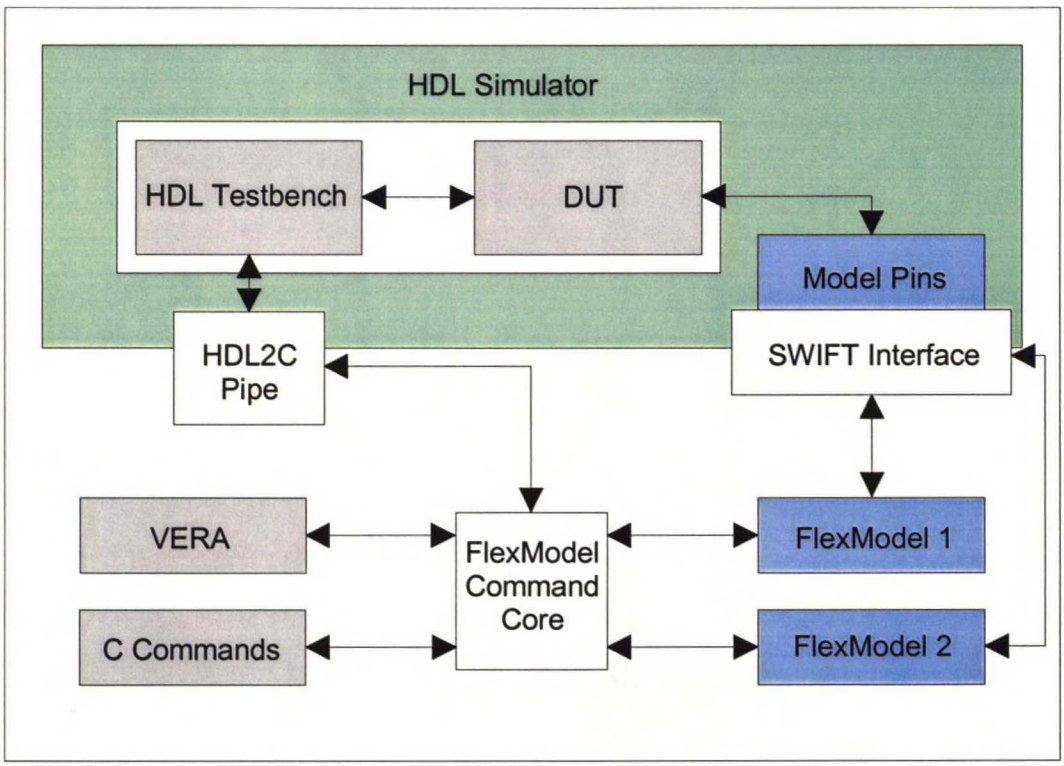


Figure 12 FlexModel structure and interface [22].

FlexModel structure and interfaces are explained in Figure 12. In HDL simulation, there is a testbench which includes the device under test and stimulus. If the FlexModel is controlled with VHDL or Verilog, these commands are also in the testbench. The

FlexModel command core controls the FlexModels and model commands have an interface to it. HDL commands are connected through HDL2C Pipe and other command methods connect directly to FlexModel Command Core. The model pins are connected through SWIFT interface to the actual simulation. This means that when the FlexModel Command Core receives a command, it issues it to the corresponding FlexModel which then drives the model pins through the SWIFT interface to a desired state.

Connection to simulators can be done the same way as in SmartModels. The easiest way to connect FlexModels to simulator is to use the HDL testbench example which is included in every model. This is connected to a HDL based simulation or in QuickSim Pro the VHDL entity is connected to the symbol. FlexModels have some common commands but mostly commands are model specific. Processor models have commands like burst write and burst read. FlexModels support a feature that returns the read data to HDL command stream so you can make a comparison to written data and determine if it was correct.

The commands in HDL command stream are read to the FlexModel Command Core and executed in the same order they were received. It is also possible to execute one command and after that one is executed read the next command. This way there could be some checks after the first command is executed to determine which command will be executed next. Every instance in simulation could have different standard timing file i.e. same models in simulation can behave differently in the design. User defined timing is possible to components only, which means that all models of a specific component use the same user defined timing file in the simulation. FlexModels run up to 40 % faster in function-only mode than in timing mode [22].

SmartModel library includes also various types of bus interface models. These models are FlexModels and they are controlled the same way as other FlexModels. Bus interface models create a virtual system around the design under test. One example of bus interface models is the PCI bus model. This model is used to test devices connected to the PCI bus against the PCI specification. The model has the same functionality as the PCI bus and timing violation checks are also done. PCI cycles are created with commands like memory write and memory read.

3.3.3 Memory Models

Memory models can be created with tools that are designed for this purpose. Synopsys has a memory model tool called MemPro and Denali Software has a tool called MemMaker. These both tools have a similar graphical user interface where the memory specifications are given. The tool generates the memory model automatically from the specification to a desired simulator. Both tools can create memory models in VHDL or Verilog format. These tools support various types of memories like EEPROM, SDRAM, SRAM and DDR SDRAM. Models can be created for System on Chip design or to System on Board design verification. MemMaker and MemPro can also be used to create models for FIFOs.

The models can be created manually by selecting the type of the memory and then data and address bus widths and other parameters. Timing information can also be included and the tools use similar notation than the component vendors use in their data sheet. Easier way to create memory models is to download a memory specification file from the web. Denali Software has developed a Specification of Memory Architecture (SOMA) file format to describe the function and timing of the memory model. Denali Software maintains a database of over 3000 SOMA files [23]. The SOMA file for a certain memory model can be downloaded and used to create the memory model. This way there is no need to specify the model manually. MemPro has a similar feature and these files can be downloaded from Synopsys web site or from component vendor that supports this specification file format. The other benefit in using these memory specification files is that they are already verified by the EDA vendor. This reduces the possibility of errors in the memory models.

The memory models created with MemPro can be set to function-only mode or to timing mode. If timing violations happen, the models assert warning messages to simulator command window. The memory models can be generated to be able to hold two or four state data. This basically means that the states are 0 and 1 or 0, 1, X and U. State X means that there is a conflict for example one component is driving 0 and an other component is driving 1 to the same signal. Uninitialised state (U) means that a driver does not drive anything. Memory models can be very simple and these models would be easy to write manually using HDL. Problem is that own memory models can consume a lot of workstation memory since they create a table that can hold the whole memory data. The memory models created with MemPro use data storage dynamically which efficiently allocates the workstation memory. [24]

Both memory models support memory content loading from an external file. The contents of the memory can also be dumped to an external file during simulation. There are also debugging features where the content of memory and the operations done can be viewed. Both tools have a graphical user interface for viewing the simulation history of the memory model. This is helpful when there are problems in simulation. For example SDRAM must be initialised properly before starting to write any data to it. If something goes wrong with the initialisation, the commands can be viewed with the debug tool and the cause of the problem can be resolved.

Model format created with these tools is Verilog or VHDL as mentioned before. This means that the models are connected to HDL simulation with component instantiations. When using QuickSim Pro, the models must be connected to the schematic symbol with a VHDL entity. These models can be used only in HDL simulators. MemPro tool is able to create a simple testbench for the created model to test its basic functionality. This can be helpful when using a new memory type. Functionality of the memory can be tested with the testbench and the user sees the correct bus cycles which should be used when accessing the memory.

3.3.4 HW/SW Processor Models

In HW/SW co-simulations, special simulation models for processors are needed. Semiconductor vendors provide processor models which can be used in hardware simulators to run software code from memory. These models do not have any special features such as the Seamless environment provides. Models used in Seamless environment are manufactured by Mentor Graphics together with the semiconductor vendor or some semiconductor vendors create the models by themselves.

Motorola provides full functional and bus functional processor models which differ from each other so that the first one takes binary machine code as input stimulus. Commands to bus functional model are given through backdoor interface, which is basically a few signals that are used to give the command and other information which is needed. This means that actually the full functional model can be used to verify software and the bus functional model is used for hardware verification. These models are connected to simulators through the SWIFT interface. Full functional model reads the data from a file which includes the software code. These models that Motorola provides have a connection to the model in Verilog format and they use the same library that SmartModels do.

Seamless CVE Processor Support Packages (PSPs) supply processor models that enable accelerated co-simulation. The two main components of a PSP are the Instruction Set Simulator (ISS) and the Bus Interface Model. The Bus Interface Model simulates the processor's input and output pin behaviour for the hardware portion of the simulation. The software executes as a separate process on an Instruction Set Simulator. [25] The Bus Interface Model is instantiated to the simulation with VHDL or Verilog and it performs only the pin transactions instead of modelling the internal logic of the processor.

Mentor Graphics creates PSP models for some processor vendors. Partly the model comes from the semiconductor vendor directly. This means the peripheral ring around the processor. After the RTL code of the processor is finished, the PSP model creation can be started. The RTL code is modified so that the intellectual property is safe. This code is delivered to Mentor Graphics which then develops and verifies the model to be correct. [26] As described in Figure 10, the RTL code is ready early in the development phase. This means that the PSP model can be ready before the actual chip is manufactured. When designing a new product, this can be very helpful when time to market is critical.

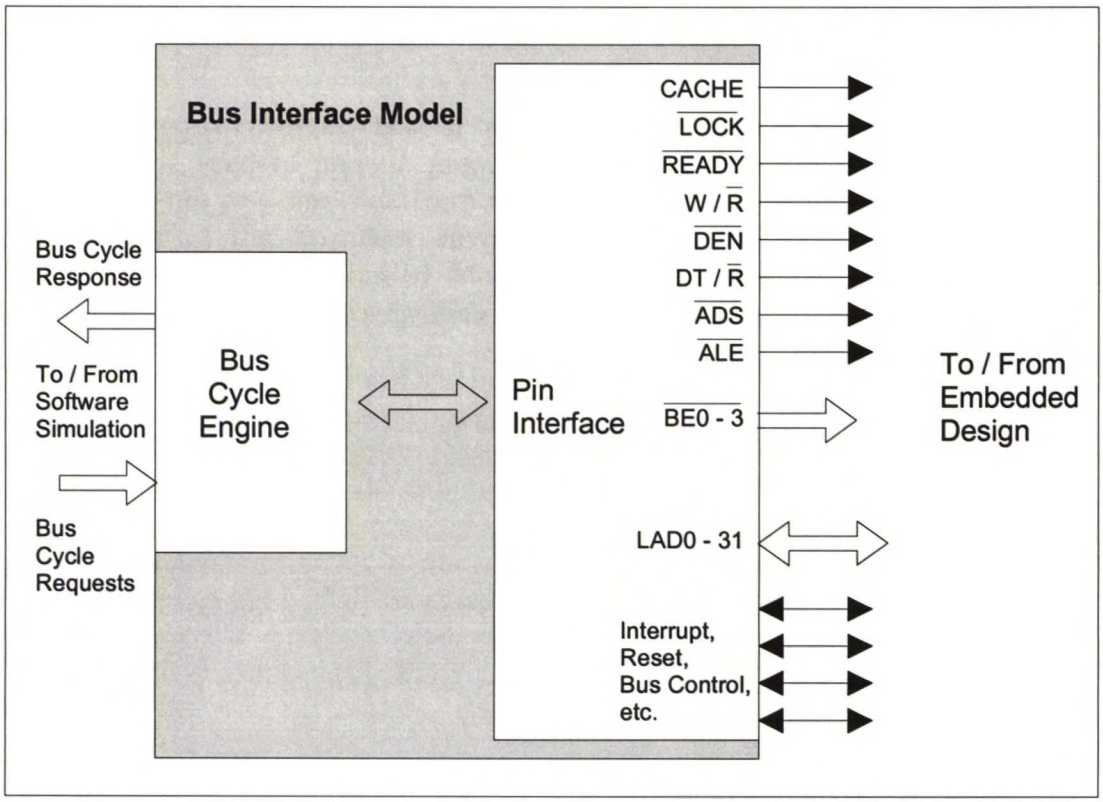


Figure 13 Bus interface model [25].

Bus Interface Model is shown in Figure 13 which clarifies how the model works. The bus cycle requests come from the ISS, which runs the software. Bus Cycle Engine performs the logical operation through the Pin Interface, which is connected to the design. When the Pin Interface receives a request, the Bus Cycle Engine communicates the request to the ISS. This kind of request could be for example an interrupt. Data access request through the Pin Interface can be cycle accurate depending on the processor clock or in some processor models also timing is possible.

Machine code can be run on the Instruction Set Simulator. Another way of running software on the PSP model is using Host Code Execution (HCE). This is a technique where the software is compiled and run directly on the workstation. One advantage is that the code runs much faster than machine code on ISS. Another advantage is that it allows portions of software to be implemented in high-level language like C and simulated in hardware without having the machine code available. The source code must include calls to an Application Program Interface (API) which links it to the Seamless CVE object Library that contains the API. Compiled code is then executable on workstation and it connects to the Bus Interface Model in the hardware simulator. [25]

3.4 HDL Write

HDL Write is a Mentor Graphics program which converts schematics to a VHDL or Verilog netlist. This allows the schematic connections to be simulated with an HDL

simulator. This same program can be used in connecting VHDL models to schematic symbols. A VHDL entity can be created automatically from a schematic symbol which can be used to connect the simulation model to the symbol. QuickSim Pro uses this tool also when it connects the HDL parts of the design to ModelSim. It actually writes a VHDL netlist of the design and uses that part in ModelSim.

The tool creates a collection of files which contain the actual VHDL code. Every schematic sheet is written to a separate file. Hierarchy of the design is modelled with components. Every schematic symbol is written as a component in VHDL and these are then connected together. This means that the hierarchy looks similar in the simulator as in the schematic sheets. The tool is controlled with an options file which is used to give parameters such as: is the component instantiation done in the same file it is used or in a separate file. There are also VHDL parameters which can be added to schematics to provide the tool more information. This makes the code more readable as one can name internal nets with names that describe the purpose of the net. Also components can have a name property which is used in the VHDL code. This makes it easier to find a specific component when there are several components of the same type. [27]

SmartModels, which are connected to the schematic symbol, are instantiated to the netlist as components. These connect to the actual SmartModel with a wrapper which was described in chapter 3.3.2. Other simulation models are connected manually to the component which is created from the symbol. If the symbols in the schematics contain VHDL information, the tool can automatically connect the models to the netlist. Thus the HDL model used in QuickSim Pro simulations are connected automatically.

Port types are determined from the schematics directly. If a bus is drawn with a busin symbol, it is an input port in the VHDL code. In real products it does not matter which type of bus is drawn to schematics, because the bus will be wires on PWB. This tool, however, will not write functionally correct code if the port types are incorrect. The schematics must be checked and edited before using the tool to be able to get correct VHDL code which will actually work.

4 SIMULATION

4.1 Simulation Flow

This chapter describes the basic plug-in unit simulation flow. Next chapters will explain how the flow is applied in different simulations. The flow is shown in Figure 14. The milestones in the Figure are from the electronics design process described in chapter 2.1. Milestones are not exact indications when the phases should be finished but they try to give some idea when the work is done compared to the electronics design process.

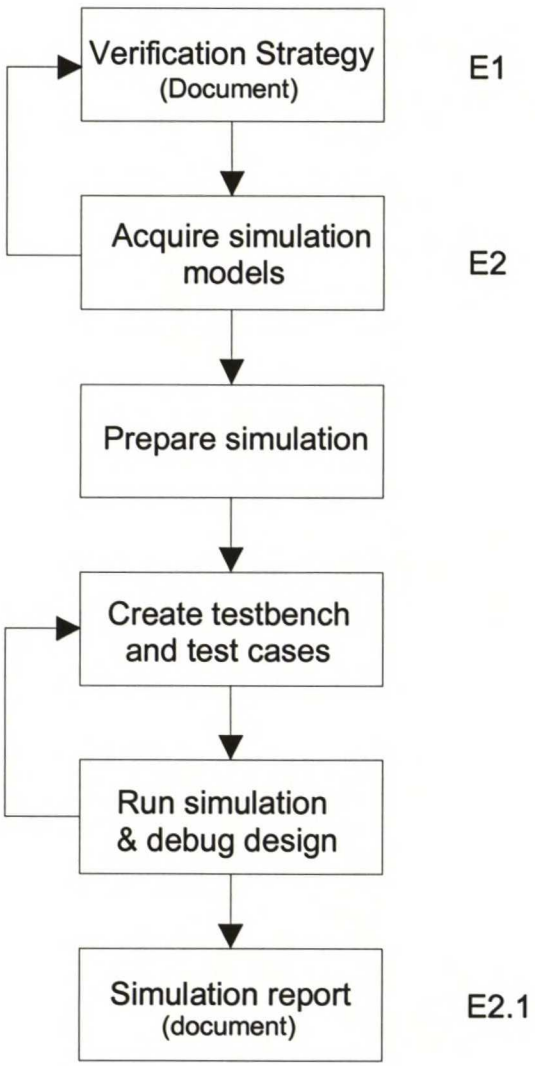


Figure 14 Simulation flow.

The first phase in simulations is the verification strategy. This phase starts with studying the design and determining which parts can and should be simulated. Also the simulator

which is used in simulations should be determined. Designs can consist of blocks that are used in other designs and proven to be functionally correct. These obviously are not the parts that should be simulated. Verification strategy can be written after the implementation specification of the design is ready at milestone E2. In Figure 14 the milestones mean that the work with the verification strategy must be started during the writing of the implementation specification. After determining which parts should be simulated, the acquiring of the models starts.

Models can be found from the SmartModel library directly. If the component is new, models are not necessarily updated to the library that Nokia has. All SmartModels can be found at the Synopsys web site and updated to Nokia's library. It is possible to request a model to be added to the SmartModel library but this can take too much time. This means that the components should be known very early in the development process. In reality this is usually not the case. Simulation models can be found directly from component vendors. Unfortunately many vendors do not have models suitable for functional simulation on their web site. This is due to the fact that these models could reveal the internal function of the circuit which obviously is not what the component vendor wants. There are a few component vendors which use model compilers to hide the internal functionality of the circuit. These will compile the HDL to binary which is impossible or at least very difficult to convert to the actual source code. If the model is not found from the SmartModel library or the component vendor, the last option is to write the model. The feasibility of this must be studied in every case. It is not very useful to model a complex circuit because it takes too much time and the quality of the model would not be very good.

After the models are acquired, the verification strategy must be re-evaluated. If some models are not found or their functionality is limited, the parts that are simulated must be determined again with the knowledge of what kind of models are available. This process is iterated until the strategy is based on the models that are available. In some cases it could become obvious that the simulations cannot be done. This could be due to the fact that models are not found or their functionality is so limited that simulations will not give any added value to design process.

Preparation of the simulation starts with checking that all tools used in simulation are up to date. There are many software releases every year and these can contain features that are relevant to the simulations. This should be done when the software is released but it is good to check that relevant software versions are installed before starting the preparation of the simulations. This phase includes also preparation of the schematics for the simulations. If we look back to the Figure 1, we can see that the schematics are ready little before milestone E2.1. The schematics should be at least almost ready before starting the preparations. This is because there is no point in doing the same thing several times when modifications to the schematics are done. There are usually parts or blocks which are ready so these should be prepared at first. Models can be prepared also immediately when they are acquired. This way, when the schematics are ready and prepared for the simulation, models are already usable. Depending on the chosen simulator the schematics are converted to VHDL or they are used directly with

QuickSim Pro. In both cases some modifications are needed and the models have to be prepared.

Testbench and test cases are created in the next phase. Depending on the simulator chosen the testbench is a little different. Testbenches used in functional simulations are described in the next chapter. Test cases are preliminary decided in the verification strategy phase. In this phase these are implemented and modified if necessary. This means that when the actual implementation starts there could be some new ideas or limitations which must be discussed with the designer and changed to the verification strategy.

Next phase is to run the simulation. Usually this phase is started with debugging the simulation environment. There could be some errors in the schematics which cause the simulator to crash. After the simulation environment is correct, the actual simulations start. The test cases are run and the responds to these are checked. If necessary, corrections to the design are done and the tests are continued. After the first test cases are passed, new test cases are implemented to verify the design more. The simulations should be carried out before the PWB work starts. This way the errors in design are found before they are transferred to the PWB. It is better to find the faults earlier than in the prototype. Thus the simulations should be continued during the PWB work if the verification is not completed before the work started.

After the simulations are completed, simulation report must be written. This report describes the environment and models used in simulations. The test cases and results are also written to the report. With this report one should be able to reproduce the test cases. Simulation report is reviewed at the latest in the layout review. After this, the document is stored with other plug-in unit documentation.

4.2 Testbenches

Testbench was described in chapter 2.4 as it is used in ASIC or FPGA verification. In plug-in unit simulations the testbench is a little different. There are similarities but since the design under test is different, the testbench is also different. The main purpose of the testbench is to create the stimulus to the design under test. Usually the testbench also monitors the design's respond to a test case and reports if the test was passed or not. This feature is not necessarily needed but it helps the verification.

The structure of the testbench depends on the plug-in unit under verification. Almost every plug-in unit has a unit computer. The parts connected to the processor are verified with a processor model. This means that for example a memory is verified by giving the processor instructions to write and read from the memory. The actual test case is created in the processor model. The stimulus does not need to come outside the plug-in unit, it can also be created in the design. In this case the testbench would be the file that gives commands to the processor. To be able to use the processor models they need external signals like clocks and resets. Clocks are created in the plug-in units with oscillators.

These components are not modelled since it is easier to create the clocks with VHDL or with a force command. Resets can also be created in the same way.

If the schematics are converted to VHDL, the plug-in unit can be instantiated to a testbench which connects external stimulus to it. This top-level testbench can be used to connect several plug-in units together. In some cases, the top-level of the design could be the testbench, where external stimulus like clocks and resets are implemented. In VHDL based simulations the testbench is much more like a testbench in ASIC or FPGA design. This is due to the fact that it is fairly easy to write simple stimulus in the testbench with VHDL. This means that force commands are not needed in the simulator since the same can be written with VHDL. This makes the simulation easier to handle because it is usable when starting the simulation. If force commands were used, these would have to be assigned or at least checked every time the simulator is invoked.

Bus Interface model can also be used as a testbench. This is feasible when the plug-in unit is connected to an external bus and a model for that bus is found. Bus Interface models include VHDL testbench where the bus is modelled. The design is connected to the virtual bus in the testbench and the stimulus is created by giving instructions to the Bus Interface model. This is similar to giving commands to a processor model.

In QuickSim Pro simulations clocks and resets are implemented with force commands. More complex external data could be modelled with a component that is connected to the schematics. The inner functionality would be modelled with VHDL. Other possibility is to use scripts that would create the external stimulus. The processor models are used the same way as in VHDL based simulation since they are run in ModelSim.

Testbenches used in plug-in unit simulations could be same as in FPGA or ASIC simulations. If some external device is modelled to test the designed circuit, the same model can be used in plug-in unit simulations. Some functionality of a processor could be modelled to verify the connection to the designed circuit. If the same connection is not supported in the processor model, this part of the model could be replaced with the model created for the circuit verification. The FPGAs in the plug-in unit can be verified with the same testbench as they were verified in the design phase. The benefit of this would be that the schematic connections would also be verified. Before creating an own testbench to plug-in unit simulations, existing testbenches must be checked and determined whether they would be suitable for the simulation.

Software is the stimulus in HW/SW co-simulations. In these simulations the external stimulus like clocks and resets must be created. This is done with the same methods as in pure hardware simulations. The software replaces the commands which are written to the processor model in pure hardware simulations. This means that the HW/SW co-simulations do not need as large testbench as the same pure hardware simulation would need. The other benefit is that the stimulus is the real stimulus which will be used in the real plug-in unit. This reduces the possibility of errors made in testbench creation.

There are also special tools which can be used in testbench creation. One of these is Specman Elite designed by Verisity. The tool uses a special hardware verification

language called e. This tool can be used to create the test cases to the design. This means the external stimulus since simulation models are controlled for example with VHDL. Specman Elite can create test cases directly from the specifications or they can be created manually. It is also possible to use already implemented test cases. One example would be that you wanted to create ATM cells to test a device. Normally you would have to model the ATM cell before you can start running your test. Component vendors and Verisity have created code which can be used directly for modelling this kind of interfaces. These are similar to the Bus Interface models from Synopsys. Specman Elite is used widely in ASIC and SoC verification since their verification is very difficult. The tool can be connected to HDL simulator and it can also have connection to Seamless CVE. This connection ables the tool to run test cases depending on the software state. This way the functionality of a circuit can be tested in every software and hardware state.[28]

Testbench tools are one possibility to create test cases also in plug-in unit simulation. In SoC and ASIC verification these kinds of tools are becoming necessary, but they provide such features that are not directly usable in plug-in unit simulations. The use of these tools must be evaluated separately in every simulation case. They could be very helpful when there is a need to create external traffic to the plug-in unit like ATM cells. If there is no need for external traffic, these tools do not bring extra value to simulations.

4.3 HW Simulations

Hardware simulations mean in this Master's Thesis the simulations done with digital hardware without software as the stimulus. The main purpose of these simulations is to check the functionality of a plug-in unit i.e. that the schematics are drawn correctly. This is done by using the schematics to connect the models together to create a virtual prototype of the unit. The goal is to find the errors before the prototypes are built. When the simulation models include timing information, also timing can be checked with HW simulations. The HW simulations model the software behaviour as described in Figure 15. SW interface model in the figure refers to the commands that are given to a processor model.

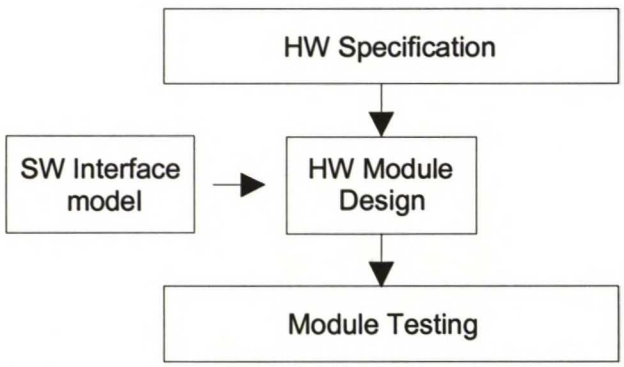


Figure 15 HW simulation in design flow.

After the simulation strategy is determined and the models are acquired, the actual work with the simulation begins. Depending on the simulator chosen the simulation preparations vary a little. In both cases the models have to be prepared to be suitable for simulations. In schematics the symbols are drawn so that they include every pin of the component. Simulation models are usually written with buses. In a schematic symbol there would be for example a 16-bit data bus that would have 16 pins where nets are connected. Simulation model of the same component would have a 16-bit wide bus. To be able to connect the model to the symbol there are two possibilities. One is to map every bit of the bus to corresponding pin on the symbol. The other possibility is to edit the schematics so that the pins are grouped to buses. This is not very wise since the purpose of the simulation was to check schematic connections so the schematics should not be edited, at least not much. The models have to be connected by manually mapping the bits on a bus to pins on a symbol, which can take a while when the components are large. This has to be done very carefully so that the errors are not done in model connection which could lead to an assumption that there is something wrong in the schematics.

When the schematics are ready, they are prepared for simulation. This means that parts that cannot be simulated are removed. These include blocks in the schematics where no models are found or analogue parts like power blocks. There are also other components like capacitors and serial resistors which have no effect on the logical function of the plug-in unit. In QuickSim Pro simulations the source voltage filter capacitors do not complicate the simulation. But in VHDL based simulation, these make the code difficult to read since they only add lines to the code without giving any function to the simulation. This is why these should be removed from the schematics before converting the schematics to VHDL. Serial resistors do not have any logical function in the simulations. They must be replaced by a net because the model of a serial resistor is a net in digital simulations. Thus it is easier to remove the resistor than to write a model and instantiate it to schematics. Pull-up and pull-down resistors can be modelled in both simulations but in QuickSim Pro it is easier to use force commands instead of modelling the resistors. There are many different resistor symbols in the schematics like single resistor symbols and resistor net symbols. This means that HDL Write makes a different component of every resistor type although their function is the same. One has to write several models, which have the same function, to be able to model every resistor in the schematics.

Plug-in units are designed so that they can be equipped with different amounts of memory. Also the processor clock frequency can be changed. There are 0 Ω resistors which are assembled to configure the function. This means that for example the memory model symbol in the schematics may not be the memory that is used in the plug-in unit at that time. The correct symbol should be changed to the schematics and the configuration resistors should be set correctly before starting the simulation. The correct configuration must be checked with the plug-in unit designer. The actual circuit can be pin compatible with several circuits but the symbols may not be. This means that the schematics have to be edited.

Port types must be correct in the schematics and in the symbols for the simulation to work. Simulators evaluate every port and determine whether it should drive data or only receive data. If the port is out or inout port, the simulation model has to drive the port at least to high impedance state because simulators expect this. Otherwise the simulator determines the value of a signal as indeterminate. This results in unexpected behaviour in the simulation. Situation like this can happen when an input port has a false port type set to inout. This kind of problem occurs usually in QuickSim Pro simulations with symbols and in VHDL simulations with schematic buses. These must be checked before starting the simulation.

After the schematics are edited and models are connected, the simulator can be started to check that everything is connected and that there are no error messages. Usually there are incorrect port types which must be corrected before getting to verify the design. If everything seems to be correct, the stimulus is prepared. This includes the force commands and the stimulus commanding the models. The creation of stimulus should be started with simple test cases. If the design involves a processor, register values are needed before the test cases can be written. FlexModel processor models need the same register values than the real device. The register values for example configure the SDRAM controller. These values are the same that are given to software designers and they are implemented to the boot software. In some cases errors in design can be found before running a complete test case when starting the simulator.

The actual simulation starts with selecting appropriate signals to waveform window and running the simulation for some time. Simulation time is usually a few microseconds which refers to the simulated time not wall clock time. Then the signals are checked and determined whether they seem correct. Clock and reset are the first signals that are checked since without these the simulation would not work. As mentioned, every signal should be in a known state. If there are signals that are in other states, the cause of this must be evaluated. It could be that something is wrong with the simulation or that the design is behaving incorrectly.

When the simulation is functioning correctly, more complex test cases can be written. These are done to ensure that every possible error in schematic connections would be found. Every bus to a device should be checked even if there is no model for a specific device. This is done to ensure that the order of bits on a bus is not reversed by mistake. This is a common error since a bus can be drawn to include for example bits from 0 to 31 or 31 downto 0. This direction must be specified in every block separately. If some block has ascending bits on a bus and another block has descending bits on the same bus. This results in a bus where looking from the both ends of the bus bit 0 equals bit 31. Writing and reading from a device does not reveal this since from one end everything looks correct. This of course applies only when the same data is retrieved like in memories.

Simulations and even the preparations are difficult to do if one does not have an understanding of the plug-in unit functionality. When the simulation strategy is made, the basic functionality of the plug-in unit must be understood. Schematic and model preparation can be done with this knowledge of the design. The implementation of the

test cases needs a deeper understanding of the design's function. As the designs are large, only the plug-in unit designer has a full understanding how the design works. Since there are many different blocks, their functionality is known best by the block designer.

The last phase in the hardware simulations is the writing of the simulation report. This includes all the errors and other findings from the simulation. After the report is finished the work with the simulation is completed. When necessary, simulation can also be used during the prototype verification if it is considered to be useful.

4.4 HW/SW Co-Simulations

HW/SW co-simulations are defined in this Master's Thesis as simulations which use software as the stimulus. These simulations are used to verify both hardware and software. This is accomplished by running the software on a virtual hardware. Software can be verified sooner than in traditional design flow where the software is verified in the real prototype. The purpose of these simulations is to get the prototype plug-in unit working correctly sooner which could mean that the prototype plug-in unit boots directly.

The simulation flow introduced in the beginning of this chapter applies also to HW/SW co-simulations. In these simulations, the software must also be included in the flow. This means that verification strategy should include the needed software in the simulations. Testbench and test creation phase is not that demanding in HW/SW co-simulations since the software is used to verify the plug-in unit's functionality.

Hardware part of the HW/SW co-simulations does not change from the pure hardware simulations excluding some model changes. Special processor and memory models must be used in HW/SW co-simulations. The same steps have to be done to get these simulations working. Software is used as the stimulus so there is no need for test case creation. Clocks and resets have to be modelled like in pure hardware simulations and also if some other external data is needed this would have to be created. Software for the simulation is needed but the same software which is developed for the plug-in unit is used. This means that the software is the same but it has to be ready sooner than in traditional design method.

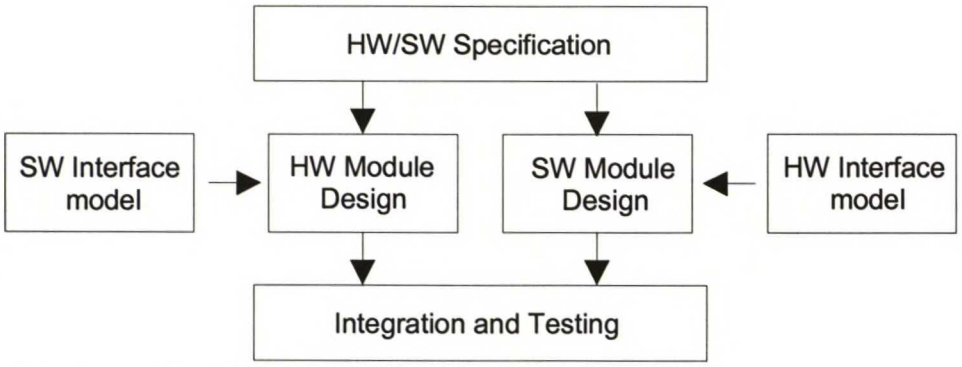


Figure 16 Traditional HW and SW design flow.

The traditional HW and SW design flow in Figure 16 describes how the software and hardware are designed and implemented separately. Requirements for the design come from both groups but the actual work is done separately. Hardware developers try to model the software with the methods described in the previous chapter. Software designers try to model the hardware with additional software. In worst case this modelling is not done on either side. Software and hardware design do not advance in parallel since the software is not needed until the prototypes are ready. In this flow, integration and testing is the first point where hardware and software meet. This means that verification is started when the prototype is ready. At this point both hardware and software is verified. Problems are usually found in both and they have to be solved before verification can continue.

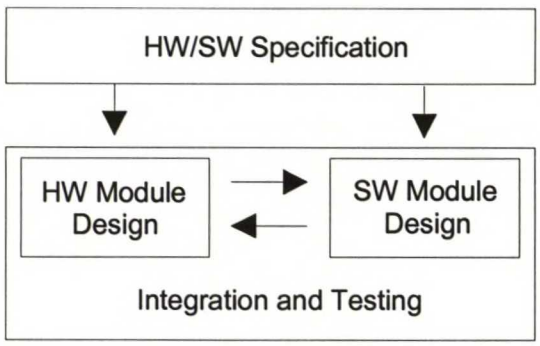


Figure 17 HW/SW co-simulation design flow.

Using HW/SW co-simulation the design flow changes as shown in Figure 17. To be able to use the real software as a stimulus, it must be ready when the hardware is designed. This means that the hardware and software design must advance in parallel. The hardware and software are verified in the design phase with running the software on a virtual hardware. The integration is done much earlier and communication between hardware and software developers is also started earlier. This way the hardware design can be changed if it is needed by the software and vice versa. In traditional flow, this is not easy to do since the hardware is ready when the software is verified.

These simulations can be used only in plug-in units which have a processor. This is due to the fact that software is run on a processor. As mentioned in the tools chapter, the amount of software that can be tested in co-simulations is determined by the simulation performance. If the software has millions of instructions, it can take millions of wall

clock seconds to execute it. In this situation the simulation takes too much time to be able to get any real value to the design process. Small parts of the software like the boot can be verified with co-simulation.

As mentioned the schematics have to be prepared the same way as in pure hardware simulations and the models must be connected to them. The models have to be compatible with the Seamless environment. At first software, which is executed in a plug-in unit, is the boot. This is stored in a FLASH memory in the real plug-in unit. In co-simulation the same FLASH image used in the real unit, is converted to Intel Hex format and loaded to the FLASH memory model. After the schematics are prepared and the models connected, the simulation can be started. Seamless CVE is invoked with the hardware simulator and software debugger. The memory content is updated to the FLASH memory model. After this, the environment is the same as in the real plug-in unit.

Simulation of the plug-in unit starts with running the boot code step by step. The processor's respond to the software is viewed from the waveform window. Boot is used to initialise the hardware so that the software can run on it. This includes for example configuration of memory controllers. The processor has to know what kind of memory is connected to it. First the processor starts to read from a specific address and a device that is connected to a specific chip select pin. This device must have the configuration bytes stored. When these are read, the processor knows what kind of device is connected to it. Usually this device is a FLASH or EEPROM. After this the processor starts to read the boot code from the FLASH. Other memory controllers and devices are configured next. Then the boot loads the operating system and starts to run applications.

The hardware must be initialised correctly before it can be used. For example the SDRAM circuits need a mode register to be set and the memory must be refreshed in a while. These bus cycles must be checked from the waveform window to be correct and if there are some problems with the initialisation of the circuits, the models will give warning messages. After the boot is verified, the operating system is started. The schematic connections are also checked during the simulation when the bus cycles are examined. After the schematic connections are checked, the optimisations can be turned on and the software can be verified with larger simulation speed.

Simulation report is written at the end of HW/SW co-simulations. This report includes all hardware and software faults found in the simulation. It must also include the software versions used in the simulations. If there are some changes done to the software to be able to use it in the co-verification environment, these must also be documented. HW/SW co-simulations should be done during the design phase. Simulations can continue when the prototypes arrive if the simulation provides useful information.

5 CASE STUDIES AND RESULTS

This chapter describes three different simulation cases which were done during the writing of this Master's Thesis. All of these cases were from ongoing projects. The first two cases were hardware simulations and the third one was HW/SW co-simulation. Hardware simulations were done to plug-in units that were in design phase and an already implemented processor block was used in the HW/SW co-simulation. The purpose of the third simulation case was to evaluate the feasibility of the HW/SW co-simulation in plug-in unit design.

5.1 Case 1: Signal Processing Plug-In Unit

The first case was a new variant for a signal processing plug-in unit. This hardware simulation included only the modified blocks of the plug-in unit. QuickSim Pro simulator was used in this simulation.

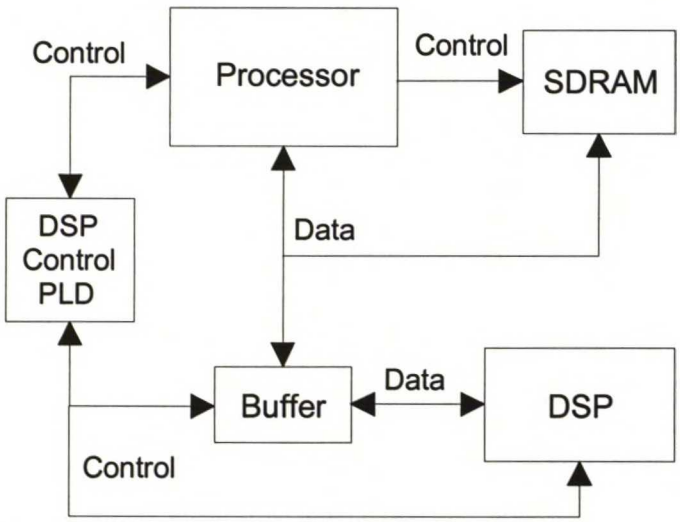


Figure 18 Block diagram of the simulated parts.

The block diagram in Figure 18 shows the parts which were simulated. In this variant the SRAM memory was replaced with SDRAM memory. Because of this change, a new buffer to DSP data bus was added. The buffer control was implemented to the DSP control PLD by modifying its design. Purpose of the simulation was to verify the SDRAM connection to the processor and the buffer connection including its control.

SmartModel library included a simulation model for the processor. The model was a FlexModel and it was controlled with VHDL. Model for the SDRAM circuit was created with the MemPro tool by inserting the parameters from the component vendor's data sheet. The buffer circuit had a full functional SmartModel connected to its schematic symbol. The PLD circuit was designed with Abel language. The SmartModel connected

to the PLD schematic symbol could use the simulation file produced by the PLDs design tool. The processor and the SDRAM simulation models were connected to the schematic symbol with VHDL.

Simulation preparations started with copying the schematics of the design and drawing a new sheet which included only the design blocks that were simulated. All serial resistors were removed from the schematics and also some other parts like test connectors were removed. Processor and SDRAM symbols were changed to the symbols where the models were connected. The simulation file was connected to the PLD SmartModel symbol. After these changes, the schematics were ready to be simulated. The stimulus was created with force commands in the simulator and by controlling the FlexModel with VHDL. This simulation needed only clocks and resets to be created externally. Other force commands were applied to simulate pull-up and pull-down resistors.

The processor model needs to be configured before it can be used to write and read from an external device. This is done by setting the same registers as in the real processor. At first the SDRAM circuit connection was verified. This was done by writing and reading from several addresses. The register settings, which configure the SDRAM circuit function, were received from the hardware block designer. Mode register is an internal register of SDRAM circuit and is set in the circuit initialisation. The value to this register is transferred automatically from the processors register settings with a bus cycle. This means that the SDRAM circuit initialisation would reveal wrong register settings in the processor. The first register settings did not configure the circuit correctly but after these were modified together with the hardware block designer the SDRAM circuit initialisation was correct. The actual SDRAM verification was done by comparing written and read data.

SDRAM circuit uses two addresses, row and column address, to point to a data. The processor gives at first the row address and then the column address. When writing several times the processor can give the row address only once and then give the column addresses with the data. This way the processor can write faster to the memory. The row is left open until it is closed with a precharge command. SDRAM circuits support writing with auto-precharge which means that the memory automatically closes the row when the writing is completed. This simulation revealed a bug in the processor model because it was writing with auto-precharge to the memory when it should have written without auto-precharge. When data was written to SDRAM, the row was closed after the writing was completed. If another write cycle was issued to the same row, the processor model assumed that the row was open and gave only a column address. The memory model did not accept this and gave an error message. This was not a severe bug since with a force command, the auto-precharge can be disabled and the design verification could be continued.

The SDRAM simulation did not reveal any incorrect functionality. There was a minor error in the SDRAM circuit DQM pin connections, which are used to enable the data output. One DQM pin controls one byte on the output pins. The data bus coming from the processor was inverted when it was connected to the memory but the DQM pins

connection was not inverted. This error was discovered by the plug-in unit designer and it was verified with the simulation.

The other modified block included the new buffer and the changed PLD design. This simulation was done by writing and reading from the DSP which actually was not modelled. Data written from the processor was examined from the waveform window after the buffer circuit. When read cycles were used, the data was created using force commands. This was considered to be sufficient verification of the schematic connection and the PLD design. The purpose of the PLD was to control the data direction and the output enable of the buffer.

This simulation did not reveal any errors in the design. The register settings and the incorrect DQM pin connection were verified. Although there were no actual errors found, the simulation supported the design process and the prototype unit had no hardware related problems. This proves that the simulation results were valid.

5.2 Case 2: Message Bus Plug-In Unit

The second case study was done to a new variant of the message bus plug-in unit. This plug-in unit is part of Nokia's older DX 200 platform, which is similar to IPA 2800 platform when the interest is in simulations. Message bus plug-in unit provides a message interface between computer units. As a simulation case, this was a good example of hardware simulations. This simulation was done by converting the schematics to VHDL and simulating the design with ModelSim.

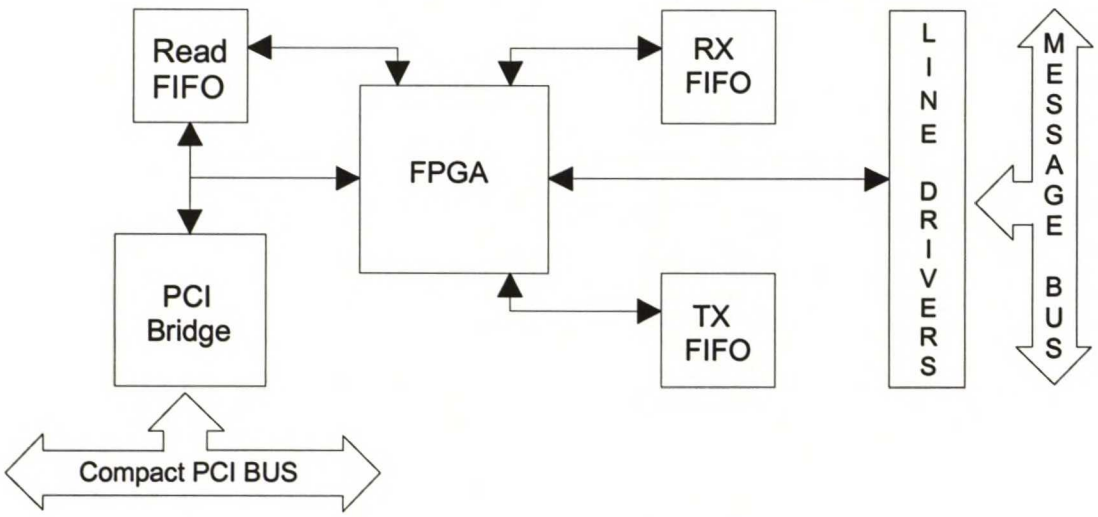


Figure 19 Block diagram of the message bus plug-in unit.

The block diagram and the connections of the message bus plug-in unit are shown in Figure 19. The plug-in unit is connected to the computer unit with a Compact PCI bus. Message bus connects the message bus plug-in units together providing a

communication interface between the computer units. This means that the plug-in unit can be considered as a network interface card.

This new variant included only one unchanged hardware block from the previous variant which was the PCI block. This meant that there were a lot of changes that needed to be simulated. The whole plug-in unit function is controlled with an FPGA and it uses FIFOs to store the messages. Connection to the message bus is implemented with line drivers and receivers. The last functional block includes the PCI bridge which provides an interface to the Compact PCI bus.

The computer unit controls the message bus plug-in unit through to the Compact PCI bus by giving predefined commands. When a computer unit has to give a message to another computer unit, it writes the message to the message bus plug-in unit and gives a send command. Then the message is delivered through the message bus to another message bus plug-in unit. Now the unit computer, which should receive the message, can read it from the message bus plug-in unit it is connected to.

The functionality of the plug-in unit is implemented in the FPGA which was designed during the plug-in unit design process. Also the PCI bridge was originally partly designed in Nokia which means that the VHDL code for that circuit was available. RTL VHDL code was used in both of these main components to simulate their functionality. Since the computer unit connects with the Compact PCI bus to the message bus plug-in unit, a PCI bus interface model was needed. This was included in the SmartModel library. The FIFO circuits' specification files were downloaded from Synopsys web site and MemPro tool was used to build the models. Line driver and receiver circuit had no functional simulation model available so the model had to be created with VHDL. This simulation model was described earlier in chapter 3.3.1. Pull-up and pull-down resistor models were also implemented with VHDL. Other simulation models, which included basic logic gates, were modelled with SmartModels.

Verification of the plug-in unit started by verifying that a message could be sent with one message bus plug-in unit. After this was done, a few plug-in units were connected together and messages were sent between them. This kind of connection is easier to do with VHDL than with schematics. So the schematics were converted to VHDL. The first attempt was to convert the schematics to VHDL without modifying them, but this revealed to be very time consuming since the VHDL files had to be edited manually. Other problem was that the tool did not make right connections if port types did not match. The FPGA symbol had all ports set to inout but the buses coming into the schematic sheet had in, out or inout data direction. HDL write left ports, which port types did not match, unconnected. With a lot of editing, the schematics were finally converted to VHDL.

Simulation models were connected to the VHDL converted from the schematics. The PCI bus interface model was problematic to connect since the PCI connector was not on the top-level schematic sheet which would mean that it would be on the top-level in VHDL. There were two possibilities. One was to manually transfer the PCI connector signals from the lower hierarchical level to the top-level. The other possible solution and

the one that was implemented was to make a component of the PCI interface model and instantiate it to the design. Usually the external stimulus is connected through the top-level testbench.

Next step was to write the first test case. The stimulus, which controlled the PCI bus interface model, was written with VHDL. The first test case included the PCI bridge initialisation and a message which was sent to the plug-in unit itself. There were some configurations which had to be made before the plug-in unit was functional like giving the unit address. These were done by setting values to signals with VHDL. Clocks were created by writing a VHDL process that created the clocks. The 33 MHz PCI clock was automatically created in the PCI bus interface model and also the PCI reset came from the same model. After these were done, the simulation could be started.

As mentioned before, the FPGA was developed during the plug-in unit design process. According to this process the FPGA design has to be ready when the prototype plug-in units arrive. These simulations were carried out to verify the schematic connections before the PWB work started. This means that the RTL code of the FPGA was not the final version and it was modified many times during these simulations.

The first test case did reveal many faults which were mostly in the FPGA VHDL code. At first the message must go through the PCI bridge to the FPGA and from there to the TX FIFO. When a send command is given, the FPGA sends the message from the TX FIFO to the message bus. The message is received by the FPGA from message bus and stored into RX FIFO. From there it is moved to READ FIFO where the computer unit reads it. The debugging started by checking from the waveform window where the message data was transferred. The first bug was found from the line driver and receiver model. The model drove Z to receiver port if there were no data on the differential signals. This was incorrect since the actual circuit drives 0 when the differential signals are not driven i.e. they have indeterminate values. The FPGA assumed that the value would be zero thus it did not function correctly. A second bug was found from the READ FIFO reset signal. This signal was incorrectly inverted twice resulting in a situation where the FIFO was in reset when other circuits are not. After correcting this error the message was sent and received correctly.

The bus interface model compared the read data automatically to the expected data. If these did not match, it gave a warning to the simulator command window. This feature was helpful since now the correct response was not only visually checked from the waveform window. Next step was to increase the number of written messages. The plug-in unit can handle more than one message at a time by storing them into the FIFOs. Three messages were written and sent before any messages would be read. This tested the functionality of the plug-in unit very well since it had to store the messages correctly to read FIFO and leave one message to the RX FIFO. After some modification to the FPGA VHDL code was done, this test case was correctly executed.

To this point all test cases were done with one message bus plug-in unit which sent messages to itself. This simulation reveals schematic errors in one plug-in unit but when there are several units connected together some new errors could be revealed. The

VHDL of the message bus plug-in unit was modified so that it was a component which had message bus signals and some other configuration signals as ports. A new testbench was created which included three message bus plug-in units and the message bus between them.

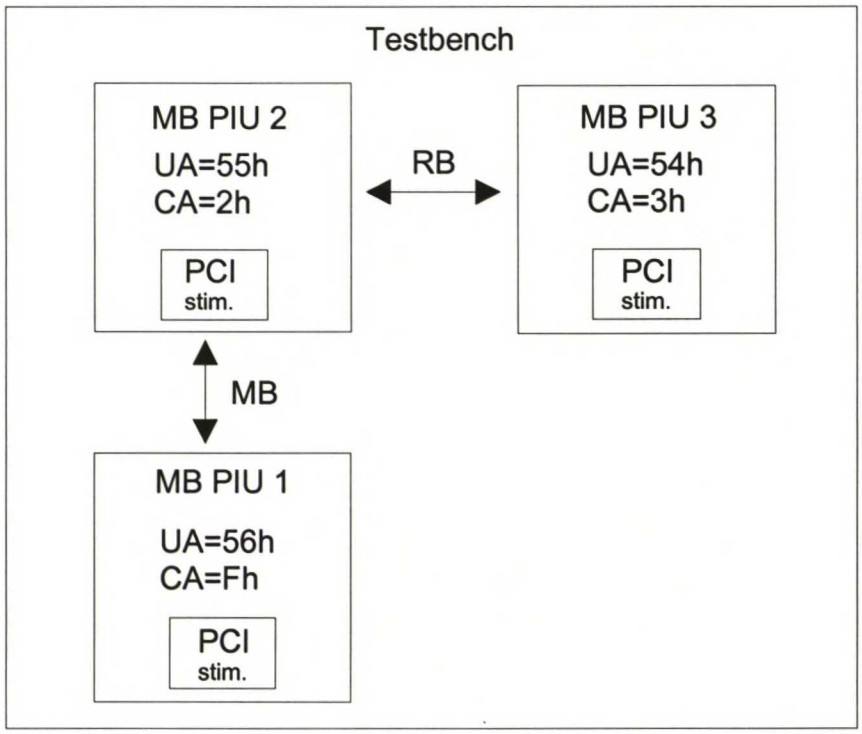


Figure 20 Simulation configuration.

The message bus is divided into segments. One cabinet has a message bus (MB) and the cabinets are connected together with a repeater bus (RB). This way the physical length of the message bus is shorter compared to a bus without segments. The data is the same in every segment and in the repeater bus. The second simulation configuration is described in Figure 20 where the connections can be seen. The message bus plug-in units were configured to have different unit (UA) and cabinet addresses (CA). Every simulated unit had a separate PCI bus interface model connected i.e. a separate Compact PCI bus.

Test cases done in the second simulation configuration included sending messages between the message bus plug-in units. A message was written and sent from MB PIU 1 with address 54h. This message was then read from the MB PIU 3 with the PCI bus interface model. The message had to be transferred correctly from the message bus segment to the repeater bus and to other message bus segment. Message bus plug-in unit always reads the message from the message bus although it is a unit that is connected to the repeater bus. Thus the configuration in Figure 20 verifies the whole message path. This test revealed some bugs in the FPGA source code but the schematic connections were correct. Other test performed with this configuration was command message sending. The computer unit can also send command messages which are a little different

from normal messages. This test revealed also errors only in the FPGA source code. After correcting the FPGA source code, all tests were passed.

This simulation revealed also two faults in the schematics before the simulation was even started. These were found during the schematic conversion to VHDL and model connection. One fault was that a wrong bus was going to the test connector. This was a minor fault but helped the prototype debugging. Other fault was that bit 13 going to the line drivers and receivers was ripped twice from a bus and bit 12 was not ripped at all. Simulation would also have revealed this error but now it was discovered sooner.

Converting schematics to VHDL proved to be more difficult than expected. The tool does not write correct code if the schematics are not drawn the way the HDL write tool expects. For example port types have to be correct to use this tool but PWB tools work fine with incorrect port types. These difficulties showed that the schematics should be drawn a little differently and some parts like serial resistors should be removed by editing the schematics. This will help editing of the VHDL files a lot. Despite the large amount of work required to convert the schematics to VHDL the simulation was much easier to control than a schematic based simulation. Now there is only one user interface and it is not necessary to use force commands since these can be written directly to VHDL. This helps the simulation control even more. It was assumed that it is much harder to understand the VHDL code of the schematics than the schematics seen directly in QuickSim Pro. Finding a bus from the VHDL code is a little more difficult but everything is still quite easily found. After VHDL properties are added to schematics, the use of VHDL instead of schematics does not reduce the visibility of the design compared to QuickSim Pro simulations. Other benefit of using the HDL write tool is that one has to read the schematics very carefully.

Message bus plug-in unit simulation revealed a few faults in the design. Especially the reset signal inversion might have caused problems in the prototype verification. The FPGA VHDL code was developed a lot during these simulations. Many bugs were found which helped the FPGA verification. Some bugs might have even been missed in the FPGA simulations. Now the whole plug-in unit functionality was simulated which was more than the FPGA simulation would cover.

Plug-in unit simulations were done together with the FPGA designer. This proved to be very helpful since most of the problems encountered were FPGA related. It would have taken a lot more time to understand how the FPGA should work without the help of the FPGA designer. Now the FPGA designer could tell what was wrong and make a correction to the VHDL code. The plug-in unit designer was also involved in the simulations but more as a consultant when schematic errors were found.

After the simulations were completed, one error in the schematics was found. The problem was with the connectors which are used to connect the plug-in unit to the message bus and the repeater bus. This connector was not included in the simulation because it would have taken a lot of effort to connect plug-in units together using this connector. Instead the buses going to the connector where connected together. This proved that parts left out of the simulation should be selected very carefully. Here the

selection was made because it would have taken additional time to model the connector and the value from it was considered to be minimal. This obviously was a wrong assumption.

5.3 Case 3: Computer Core Hardware Block

Purpose of this simulation case was to study the possibility to use HW/SW co-simulations in plug-in units designed in Nokia. Since the nature of this simulation was to evaluate the tools and the method, an already verified processor block was used. This way the design and the software were both correctly implemented. If there would be any problems, they would have to be caused by the tools. This simulation was limited only to the processor block of the plug-in unit. The goal was to study the feasibility of HW/SW co-simulation with running the boot software and start up hardware verification program.

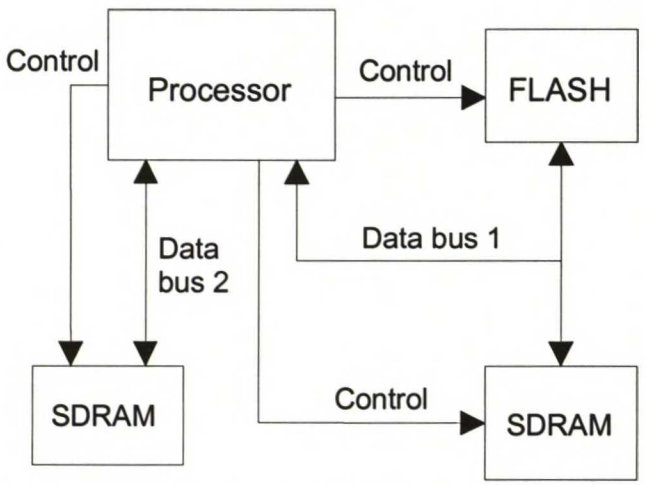


Figure 21 Processor block used in HW/SW co-simulations.

The processor block used in this simulation case is shown in Figure 21. This same block is used in several IPA 2800 plug-in units. HW/SW co-simulations included only the processor and memories. The boot is executed from a FLASH memory and other memories are SDRAM in this design. There are two different data buses in the processor where the memories are connected. The FLASH memory is in the same bus as the other SDRAM memory. Resets and clocks are the only external stimulus needed.

The tools used in this simulation were Seamless CVE, ModelSim and XRAY. Since this was HW/SW co-simulation, a special processor model was needed. It was acquired from Mentor Graphics. The memory models were created with Denali MemMaker tool by downloading the SOMA files from the Denali web page. These models were chosen to be able to use Seamless optimisations. XRAY debugger came with the PSP. All simulation models had HDL connection to simulation which meant that ModelSim had to be used. Due to the fact that these simulations run quite slowly, logical choice was to use only ModelSim instead of QuickSim Pro.

The schematics had to be converted to VHDL before the models could be connected. This was done the same way as in case 2. A top-level testbench was also created where clocks and resets were implemented. There were similar problems with port types as in the previous case. After correcting these port types, the simulation environment was functioning properly. Next step was to get the right software. In Nokia's case, the software is in the FLASH memory. A memory image was created and it was converted to Intel HEX format which can be loaded through the Seamless CVE user interface to the memory image server.

At first the Seamless CVE must be invoked. The second step is to invoke the hardware simulator and software debugger. The software debugger connects to the hardware simulator when the processor's power up reset is deasserted. At this point it loads the executed software. The software can also be updated to FLASH memory in Intel HEX format. This models what happens in the actual device since the software is programmed to the FLASH memory. After the connection between software debugger and hardware simulator is established, the simulation is mainly controlled from the software debugger.

Everything was set-up for the software to be executed but the PSP did not start the execution. This was because it did not complete the reset cycle. This was a difficult problem to solve, but since the actual device works, the problem was in the PSP. A new version of the model was received which did not have this problem and the software execution could be started. It was observed that the simulation speed was quite slow, but with optimisations turned on, the code could be executed little faster.

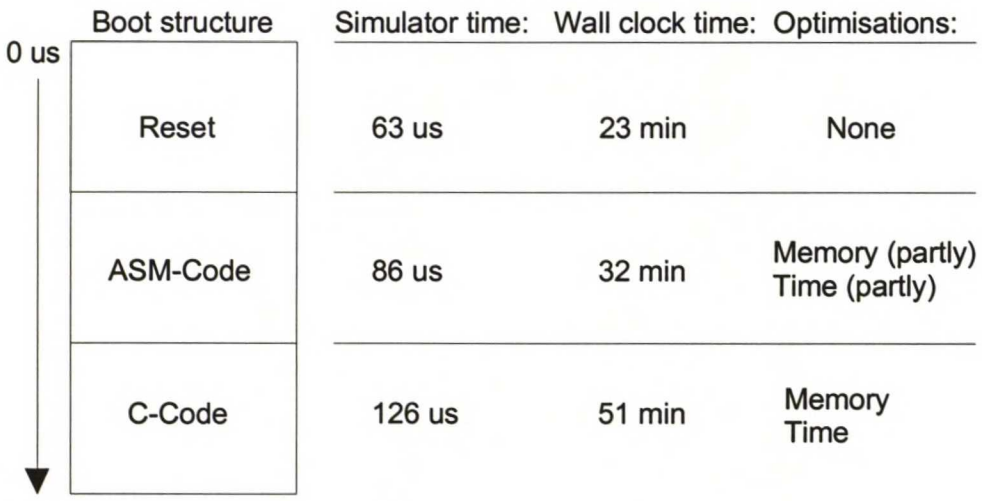


Figure 22 Boot structure and execution times.

Part of the boot structure and the execution times are described in Figure 22. The wall clock execution times include also the time consumed in controlling the tools. This means that the wall clock time does not give exact information about the hardware simulator performance. The reset sequence of the processor model takes a long time. So before the actual code is executed, 23 minutes has passed. This means that the simulation is not easily started when some errors are found. The state of the simulation cannot be saved so the simulation must be run from the beginning every time. At the beginning optimisations do not increase the speed of the simulation since the reset

sequence must run in the hardware simulator. Next part in the boot is the ASM-code which initialises the processor partly. Here the memory and time optimisations can be used partly. FLASH memory can be optimised the whole time, but the time and SDRAM memory optimisations can be turned on after the SDRAM circuits are initialised. C-code is the last phase before starting the operating system or hardware verification program. All optimisations can be on during the execution and when the operating system or hardware verification program starts its initialisation, 51 minutes wall clock time has passed. The long reset sequence takes a big portion of the whole execution time. Mentor Graphics is trying to change the model so that the reset would not be so long.

When the first part of the boot is executed, the operating system or hardware verification program is copied from the FLASH to SDRAM memory. This copying did not work because no data was correctly copied to the memory. At first it was discovered that the address sequence of the SDRAM has to be given to Seamless in simulation start up. SDRAM circuits store data to different banks using row and column address. At first row and bank addresses are given and then column address. Memory controllers split the processor's logical address to these addresses. Since Seamless uses the memory image server, it must know where to store the data coming from the SDRAM circuit in the logic hardware simulator. The address sequence gives Seamless the information which bits in the logical address correspond to row, column and bank addresses. This way, when the SDRAM circuit is accessed, Seamless can construct the logical address from the row, column and bank addresses. In real devices this information is not relevant, but when optimisations are used, the data must be stored to the right logical address with hardware and optimised access. Seamless has a macro called check_mem_config which is used to test memory operations. It writes data with hardware access and then reads it back with optimised access. If the address sequence is correct, the data is stored to the correct place in the memory image server and the optimised access returns correct data. The second check is done by writing with optimised access and then reading with hardware. This reveals if the data is stored correctly. Unfortunately, the address sequence did not correct the problem with memory copy. The cause of this problem was cache which did not function properly. The cache was supposed to work in write-through mode, which means that the data in cache is copied directly to the corresponding memory. Some amount of data was seen in the software debugger memory window, but this was never transferred to the memory image server. Thus it was never copied to the corresponding memory. When the cache was disabled, the code was copied correctly and the hardware verification program started. It was tested to be functional in this environment but slow to use since it requires the hardware simulator to run.

The simulation environment is mostly a good model of the real device. Problems with the processor model were difficult to solve and showed that simulation results can be unreliable. In this case the PSP had two limitations which resulted in minor software changes but mainly the same software can be used. The code can be single stepped and the source code can be assembler or C depending on the language used in the software. This environment provides visibility to internal registers in the software debugger and to all signals in the waveform window. Memory initialisation can be viewed from the waveform window and also the memory models report if there are errors in the

initialisation. One minor bug was found in the software during these simulations. SDRAM circuits cannot be accessed before 200 us have passed from power up. There is a delay in the boot code before the SDRAM initialisation to ensure that 200 us has passed before any commands are given. This design had two SDRAM circuits and the delay loop was after the first one had already been initialised. 200 us is most likely passed already from the power up to the point where the SDRAM initialisation starts but still the delay loop should be moved before the first SDRAM initialisation.

HW/SW co-simulation provides a platform where the software can be verified before the actual hardware is ready. This also means that hardware modifications can be done, if necessary, already in the design phase. For software designers the advantage is that this environment is available weeks before the actual prototype. Because the simulation speed is so slow only limited amount of software can be verified with this environment. When new processors are taken into use, this environment would be very helpful if the PSP is available in time i.e. before the prototype plug-in units. There has to be good support from the PSP vendor during HW/SW co-simulations. These simulations proved that the processor models are not perfect, which means that bugs are found. Best would be to get the person who made the model to provide help when problems are encountered. The PSP documentation is very limited and does not provide answers to difficult problems. This means that good support must be available.

These simulations require that the software designers have the boot ready weeks before it is needed currently. These simulations require also close co-operation with different designers. The hardware designer has to draw the schematics so that they are easily converted to VHDL. One person has to have the knowledge of the tools in use. The software designer has to write the software in time and work closely with the hardware designers during the simulations. The benefit of this is that the co-operation is started earlier and both can learn from each other. Because this platform is ready weeks before the prototypes arrive, the boot can be verified and the prototype plug-in unit can boot directly. This way the development cycle is shorter compared to a situation where the boot is verified and debugged using the prototype plug-in unit.

5.4 Results from the Case Studies

The hardware simulations showed that simulation can help the design process and reveal errors in design. HW/SW co-simulation is also helpful in finding problems in both software and hardware. This means that simulations should be done to every plug-in unit that is designed using the simulation flow presented in chapter 4.1.

Simulator should be chosen according to the design in questions. QuickSim Pro is good in small designs like the one in case 1 and when the simulation models can be connected without using HDL. The problem with QuickSim Pro is that it is difficult to control because it has two user interfaces. The benefit of using QuickSim Pro is that the schematics can be used directly which results in, that it is easier to understand the nets

and buses in the design. Also modifications are much easier to do when they can be done to the schematics directly.

When using only ModelSim, the schematics must be converted to VHDL. This process is very time consuming and needs a lot of editing. Some of the same modifications have to be made also in QuickSim Pro simulations but the VHDL files need much more work. Most of this work has to be done because the symbols do not have correct VHDL properties and the schematics are not always drawn the way they should have been when using this tool. This means that with time, as the tool and the schematics develop, the process to convert schematics to VHDL will be faster. The model connection is much easier in VHDL based simulation especially with large components. Symbol of a large component can be so big that it does not fit to a schematic sheet. The schematic tools support splitting a large symbol into pieces. This makes the model connection difficult since the model has all ports and it should be connected to several symbols which have some smaller number of ports. In VHDL based simulations, these pieces can be grouped to one component and the model is connected to it. In QuickSim Pro, the model has to be connected to one symbol and the other symbol's ports have to be connected with special nets to the simulation model. This is much more difficult than in VHDL based design. Of course, it becomes also difficult in VHDL if all parts of the split symbol are not on the same schematic sheet.

Although the conversion process takes a lot of time, the simulation is much easier to control. External stimulus and configurations are easily written in VHDL. This means that force commands do not have to be used and the simulation works the same way every time it is started. When using a lot of force commands they have to be saved to script files which are then always executed at simulation start-up. Negative part is also that the visibility to the schematics is not as good as in QuickSim Pro simulations but it is still good enough. Modifications to schematics are also more difficult to do now that they have to be written with VHDL. Despite these negative facts the VHDL based simulation is better in larger designs since it is easy to configure and control. In HW/SW co-simulations the benefit of using VHDL conversion is obvious since the simulation speed is the limiting factor.

Simulation models are crucial to these simulations. The above-introduced case studies were chosen so that there were models available for the components. This is not the case in all plug-in units that are designed. The problem in most cases is that the components, which are used, are so new that model vendors have not had time to develop simulation models for them. Simulation models are sometimes difficult to get from semiconductor vendors since they try to protect their designs. This does not mean that it is impossible to get simulation models but it usually takes time. These models should be acquired along with the components. If the semiconductor vendor cannot provide models, another vendor who will provide also simulation models should be chosen. Models acquired from semiconductor vendors do not usually cost anything. This is obviously a good thing but can lead to a situation where the model does not function properly or there is no support available. When models are acquired, the support that comes with them should also be discussed. The simulations described in this Master's Thesis showed that

good support is needed since the tools and the models are very complex and may not work correctly.

In some cases own simulation models can be created. These are useful when there are no simulation models or the component has for example differential signals. Thus it is not fully digital. The problem with these models is that they can behave incorrectly because the time consumed to implementation and verification is usually limited. When using simulation models from a vendor, they have verified the model behaviour and also the interpretation of the function is done by them. Own simulation models do not remove the human introduced errors described in chapter 2.4. This is why simulation models should be acquired from semiconductor vendors and model vendors when possible.

The simulations are part of the design process and the responsible person is the block or plug-in unit designer. Simulation tools are so difficult to use that it would take too much time to learn how to use them when simulations should be done. Hardware designers do not have to use simulation tools very often. This means that when they have to use them the tools have changed so much that the using of the tools has to be learned again. Solution to this is that there is at least one person in an organisation who is responsible for the simulation tools and research in this area. Every design team should have one person as a simulation responsible. This person would be responsible for the simulation set-up and the simulations together with the hardware designer. If many plug-in units are designed at the same time, there needs to be more than one simulation responsible since one person has only time to be involved for example in two plug-in unit design teams at the same time. The person responsible for simulation in the organisation can train other designers to be the simulation responsible in a design team.

The work should be distributed so that plug-in unit or block designer will write the verification strategy together with the simulation responsible. This should be done early in the design phase at the same time as the implementation specification is written. Acquiring of the models would be the simulation responsible's job when they cannot be acquired from component vendors. As said before every component should have a model when they are chosen to be used. This is not the case at the time, which means that there are components without models that are used also in new designs. The plug-in unit or block designer should acquire simulation models for new components used in design with the help of purchasing. The simulation responsible should also be involved in this process since there needs to be knowledge of the model types that will be acquired.

Simulation responsible will prepare the simulation environment. This includes the conversion and edition of the schematics, model connection and testbench creation. Schematics are read very carefully by the simulation responsible when they are edited and converted to VHDL. Some errors might be found already in this phase. Thus simulations add another schematics check in addition to reviews. This also means that the simulation responsible must have at least some knowledge of the design to be able to convert and edit the schematics. The best would be to study component data sheets and find the information from there. In reality this takes too much time and will require more resources than one person responsible for the simulations. Plug-in unit or block designer

has to explain how the design works and the simulation responsible has to read component data sheets as well. This will partly remove the human introduced errors. To fully remove them, would require too much resources.

Model connection is another task which reduces human introduced errors. Simulation responsible must check from the model's data sheet the correct connection and then connect the model to the design. If the model is not designed by the simulation responsible as was suggested, the connection of the model will also check that schematics are drawn correctly. Component symbols can also have wrong ports which will be revealed when models are connected.

Testbench creation will be the next task for the simulation responsible. The case studies showed that the first test cases should be small to test that the environment works. The simulation environment has to be verified to be correct before writing large test cases. This process can take a lot of time depending on the design. Verification of the simulation environment also verifies parts of the design. It is essential to find the source of the wrong behaviour in the simulation. There are three possibilities in hardware simulations: the fault is in the simulator, the simulation model or the design. In HW/SW co-simulations the software might also cause an error.

As the case studies revealed, also simulation models created by model vendors can have errors. No faults were found from the simulators so the correct approach would be to at first check the models and then the design. The model's functionality must be compared to the component's data sheet. Problem with simulation models is that they are usually not very well documented. This means that it is difficult to know if the model should not function exactly like the real component in the first place.

Simulations should be done together with the block or plug-in unit designer. This basically means after the simulation environment is verified but as the designs are complex there is a need to solve problems together in every phase. Usually most of the work in simulations is done during the set-up phase and the simulations are carried out more quickly. Thus the plug-in unit or block designer will get results with a lot less effort than in the traditional way where the designer has to do all the work. The interpretation of the results will be the plug-in unit or block designer's responsibility. Simulation report will be written by the simulation responsible with the help of the plug-in unit or block designer.

Simulation responsible will work as a support person in the design work, but also participate to gain the knowledge required in above-mentioned tasks. This will provide redundancy to the design work because there is another person than the designer checking the design. To gain the needed knowledge, the designer has to explain the function of the design to the simulation responsible. This means that the designer has to go over the function of the design one more time, which could reveal errors. Benefit to the designer is that there is less work to achieve simulation results. This means also that simulations are not left undone because they take too much time from the designer.

HW simulations are feasible for every design's digital part. Processor block can be verified with a simulation model that is controlled for example with VHDL. Connections to other devices can also be verified the same way. In HW simulations the limitations are the simulation models and the testbench which is written. This means basically that everything can be simulated but it takes time and resources. It has to be evaluated in every case if something needs to be simulated and how much work has to be done to achieve it. As the second case study showed, the best would be to simulate all parts that are possible to be simulated. When something is left out of the simulation, there is a risk that errors are found in that part.

HW/SW co-simulations were studied as an evaluation case which revealed that these simulations can also be used in plug-in units. Software and hardware can both be verified earlier in the design phase. This way the prototype plug-in unit can boot directly and the whole design cycle is shorter compared to a situation where the boot is verified using the prototype plug-in unit. The benefit compared to pure HW simulations is that the stimulus controlling the processor is the actual software that is used in the real product and not just commands to mimic the behaviour of the software. This means also that the extra work done to verify the design with simulation is smaller. Almost everything developed in the co-verification environment can be used in the real product. HW/SW co-simulation should be used when there are changes both in the processor block hardware and boot software. The more there are changes the bigger the benefit of using HW/SW co-simulation can be.

PSP models and Seamless CVE licenses are expensive. This is the negative side of the HW/SW co-simulations. It is very hard to calculate what would an error in the design found in this environment cost if it would be found later. The cost of an error can be very large if it is found when the product is ready. The case study showed that some bugs in the software can be found with HW/SW co-simulation. This would indicate that the money spent on licenses would be earned at least in faster development cycles but as said it is very difficult to calculate any exact cost benefit.

Although Seamless CVE and the PSP licenses are expensive, the benefit of using this environment is the simulation time. If the same simulation would be done with a full functional processor model, which is simulated fully in the logic hardware simulator, the simulation times would be much longer. In the HW/SW co-simulation case, some amount of software could be run in minutes wall clock time. Without any optimisations, same software would take hours to run. This is obviously too slow to verify any software reliably.

Seamless CVE is difficult to use and the processor models are complex. As the third case showed, support from PSP vendor must be good to be able to successfully simulate hardware and software. Simulation responsible will also need good support internally from the hardware block designer and software designer. Problems should be resolved together with the help of the PSP vendor's support. If the simulation involves a new processor, the PSP vendor should participate closely in simulations.

6 SUMMARY

The objective of this Master's Thesis was to consider how plug-in unit simulations should be carried out. The work was done to find the correct flow for the simulations and to evaluate the feasibility of simulation in current designs. Other objectives were to study which parts can be simulated and at the same time learn more about the tools used in simulation.

This Master's Thesis started with the investigation of the design processes and the environment where the simulations are done. Design processes were the basis on which the simulation flow was implemented. The tools and models used in simulations are the key elements. Without models and good tools, nothing can be simulated. It became obvious that the usage of different tools should be studied more. There are some limitations in especially HDL Write which cause problems in the simulation set-up. Models for these simulations should be acquired directly from the component vendors when possible so the lack of models would not limit the simulations.

Hardware simulations proved to be very useful as the case studies showed. These can be applied to all plug-in units designed if simulation models are available. HW/SW co-simulations can be at least applied to processor blocks of the designs. Here the limiting factor is the speed at which the simulation runs. Otherwise the environment proved to be a very accurate model of the real device excluding a couple of bugs found from the simulation model. HW/SW co-simulation should be used when there are changes to both processor block hardware and software.

Simulations should be done in close co-operation with the designers involved. The tools proved to be so difficult to use that at least one person should be responsible for the tools and help the designer to perform simulations. This also adds a redundancy check to the design process. Especially in HW/SW co-simulations, the co-operation with software designers is essential. The designs are so complex that the required competencies in hardware and software are difficult to find without co-operation. Benefit of the simulations is that errors can be found before the prototypes arrive. This helps the prototype verification and can save time and money.

Verification strategy is the first step in the simulation flow. This will be done together with the hardware designer and in HW/SW co-simulations also with the software designer. The simulation models are acquired and the environment is set-up by the simulation responsible. After this the testbench and the test cases are written. Simulations are done together with the hardware designer and in HW/SW co-simulation also with the software designer. Simulation report is written when the simulations are ready.

The result of this Master's Thesis was the simulation flow which was implemented. There is still a lot of work to do to get the simulations working the way described. The tools should be studied more and the simulation models should be acquired when

components are taken into use. Especially testbench creation tools should be examined more since they were not used in this Master's Thesis. HW/SW co-simulations need the software much earlier than it is needed currently. Support for the simulation models is needed especially in HW/SW co-simulations. Co-operation with model vendors should be increased to get better support. When all of these are done, HW and HW/SW co-simulations will be used efficiently in plug-in unit design process.

REFERENCES

1. Bergeron, Janick. Functional Verification of HDL Models. Kluwer Academic Publishers, 2000.

2. Hannus, Seppo & Louhenkilpi, Timo. Simulointi. Otapaino, 1981.

3. Kettunen, Arto. Ohjelmiston ja laitteiston yhteissimulointi. Diplomityö. Tampereen Teknillinen korkeakoulu Sähkötekniikan osasto, 1998.

4. Korhonen, Jari. Electronics Design Process Description, Standard Operation Procedure. Nokia Networks Oy, Internal publication, 2001

5. Viranko, Jyrki. Conversation, 27.2.2002.

6. Klemola, Juha. IPA 2800 Hardware, Course material. Nokia Networks Oy, Internal publication, 2001.

7. Haque, Faisal I. & Khan, Khizar A. & Michelson, Johathan. The Art of Verification with VERA. Verification Central, 2001.

8. Goering, Richard. EDA leaders trade places. Article. http://www.eedesign.com/story/OEG20011112S0017, 27.2.2002.

9. Model Technology, http://www.model.com/products/se_datasheet_new.asp, 28.2.2002.

10. QuickSim Pro Training Workbook, V8.6_4. Mentor Graphics Corporation. July 1999.

11. Valasma, Harri. HW/SW Co-Verification Training. December 2001.

12. Berg, Staffan. Co-Verification Using Seamless CVE. Presentation. Mentor Graphics Corporation. 5.12.2001.

13. Davidson, Mark. S-88.133 Digital Design with Hardware Description Languages, lecture slides, Helsinki University of technology. Autumn 2001.

14. IEEE Standard VHDL Reference Manual, The Institute of Electrical and Electronics Engineering Inc., 1988.

15. Deepak Kumar Tala. Verilog History, http://www.deeps.org/Verilog/history.html 18.02.2002.

16. Kaikkonen, Sami. Conversation, 19.2.2002.

17. ModelSim SE User's Manual, Version 5.5b. Model Technology Incorporated, 2001.

18. SN65MLVD204 High Speed Differential 50 Ω Line Driver and Receiver Data Sheet. Texas Instruments Incorporated, 2001.

19. SmartModel Library User's Manual. Synopsys Inc., October 2001.

20. SmartModel Library Supported Simulators and Platforms. Synopsys Inc., March 2002.

21. Simulator Configuration Guide. Synopsys Inc., September 2001.

22. FlexModel User's Manual. Synopsys Inc., October 2001.

23. Denali Software Inc. http://www.denalisoft.com/products_mmav_SOMA.html 21.2.2002.

24. MemPro User's Manual. Synopsys Inc., September 2000.

25. Seamless CVE User's and Reference Manual. Mentor Graphics Corporation, 2001.

26. Kumar, Ajay. Meeting, 11.4.2002.

27. HDLWrite User's and Reference Manual, v8.9_1. Mentor Graphics Corporation. 2001.

28. SoC Verification Seminars. Mentor Graphics Corporation & Verisity. September 2000.